Analysis and Optimization of Global Interconnects

Sachin Sapatnekar
ECE Department
University of Minnesota
Minneapolis, MN, USA
sachin@umn.edu

Acknowledgements

Many slides borrowed from

- **Chuck Alpert, IBM**
- **Jiang Hu, Texas A&M**
- **Prashant Saxena, Synopsys**

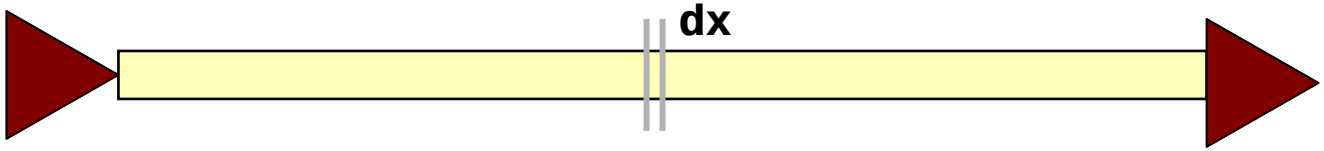
Outline of the talk

- Interconnect delay metrics
- Interconnects and scaling theory
- Synthesis of signal interconnects
- Noise and congestion issues

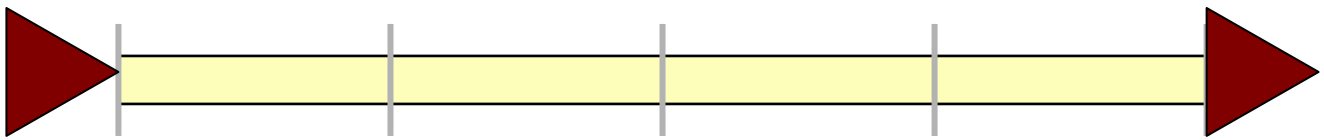
Simple delay metrics

Interconnect modeling

- Precise model requires transmission line analysis

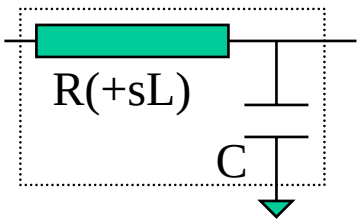


- Break up wire into segments

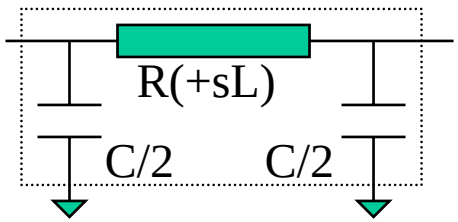


- Each segment can be modeled as

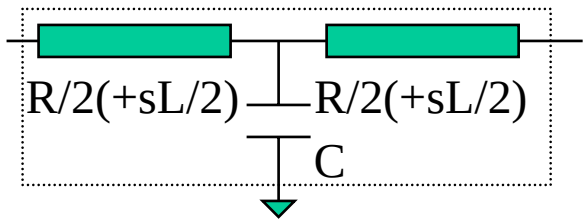
L-model



π-model



T-model



- Other issues (crosstalk etc.) modeled using coupling caps
- Interconnect extraction
  - Most precise with a 3-D field solver (takes a long time!)
  - Other faster approximate techniques useful for design analysis/optimization (R per square, C per unit area, 2.5-D models)

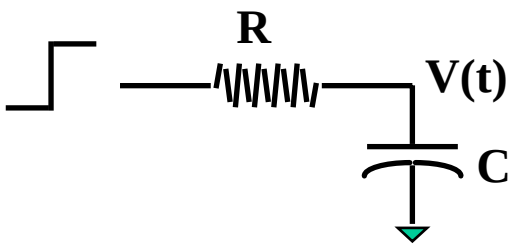
Gate delay models

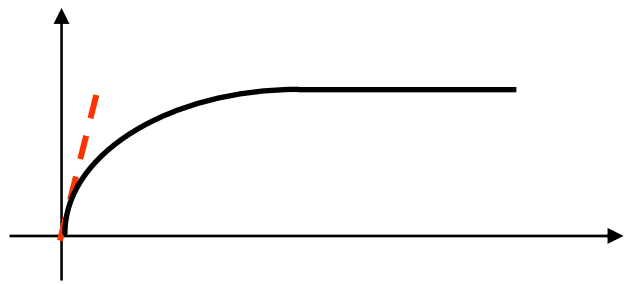
- **Traditionally: assume that the gate drives a capacitor**
  - **Build macromodels for individual gates**
    - **Delay = f(widths, transition times, loads)**
    - **Example: K-factor equations**
    - **Similar idea used in standard cell characterization:**

      Delay = f (transition times, load)
  - **Table lookup models: storage/accuracy tradeoff (e.g. .lib format)**
  - **Fast circuit simulation – used in many delay calculators**
- **More recently: effective capacitances, current source/voltage source models**

RC delay calculations

- Delays can be calculated easily
- For example: RC driven by a step excitation





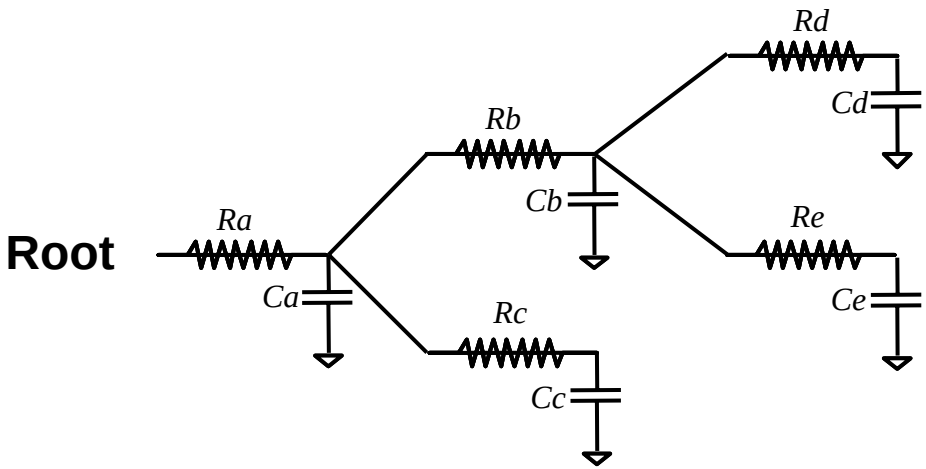
Response $V(t) = (1 - e^{-t/RC})$

Time constant = RC

Time constants for more complicated circuits?

Elmore delay for an RC tree

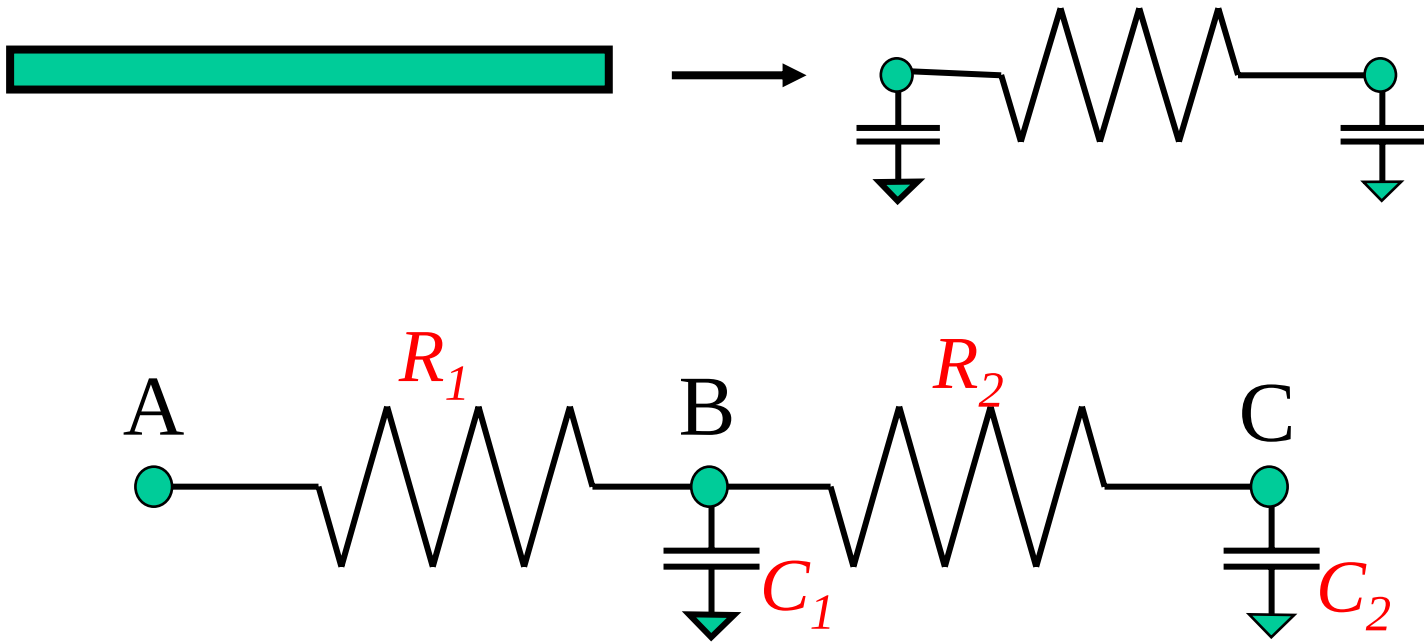
$$T_{D,k} = \sum_{i \in Path(k)} R_i \sum_{j \in downstream(i)} C_j$$



- **Elmore Delay to node e**

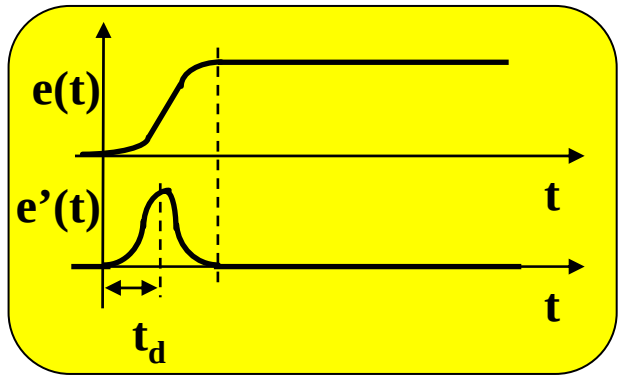
  = *Ra.(Ca+Cb+Cc+Cd+Ce) + Rb.(Cb+Cd + Ce) + Re.Ce*

Incrementally calculating the Elmore delay



$$Delay(A-C) = R_1(C_1 + C_2) + R_2C_2$$

Model order reduction methods



- **Elmore delay: RC transfer function**

$$H(s) \approx \frac{a_0}{b_0 + b_1 s}$$

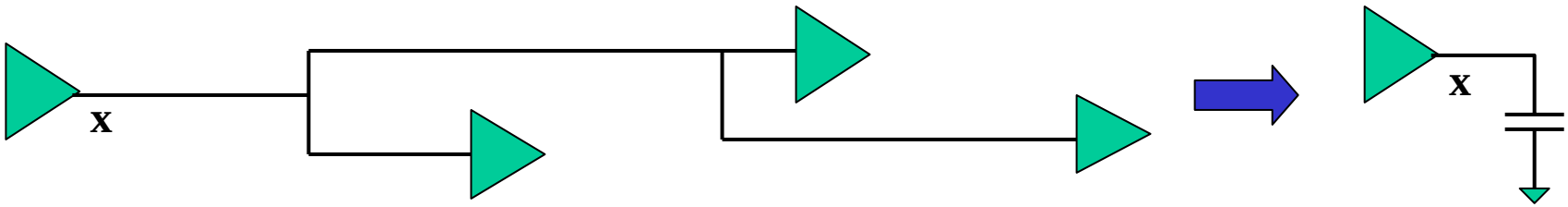
- **Can approximate RC circuit transfer function as**

$$\frac{a_0 + a_1 s + \ldots + a_{n-1} s^{n-1}}{b_0 + b_1 s + \ldots + b_{n-1} s^{n-1} + b_n s^n}$$

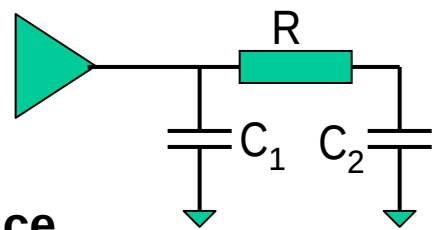
  - Response approximated as a sum of exponentials
  - Useful for interconnect simulation
  - Other variants: PVL, PRIMA, etc.
  - Handles linear systems, but drivers may be nonlinear

Effective capacitance model

- Includes the effects of gate nonlinearities
- Gate driving RC interconnect

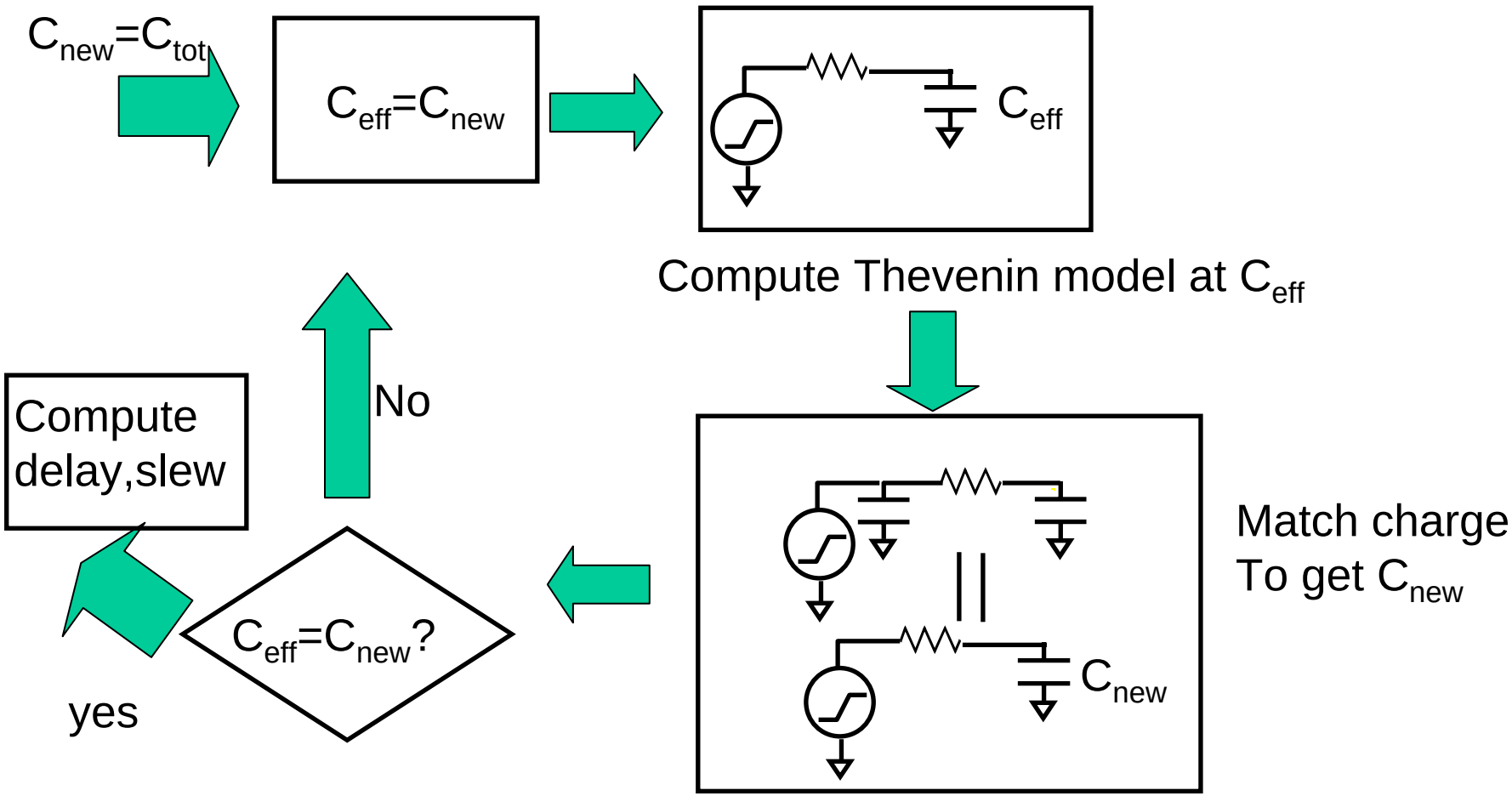


  - Determine waveform at gate output; analyze interconnect as a linear system after that
- Possible model for waveform at x



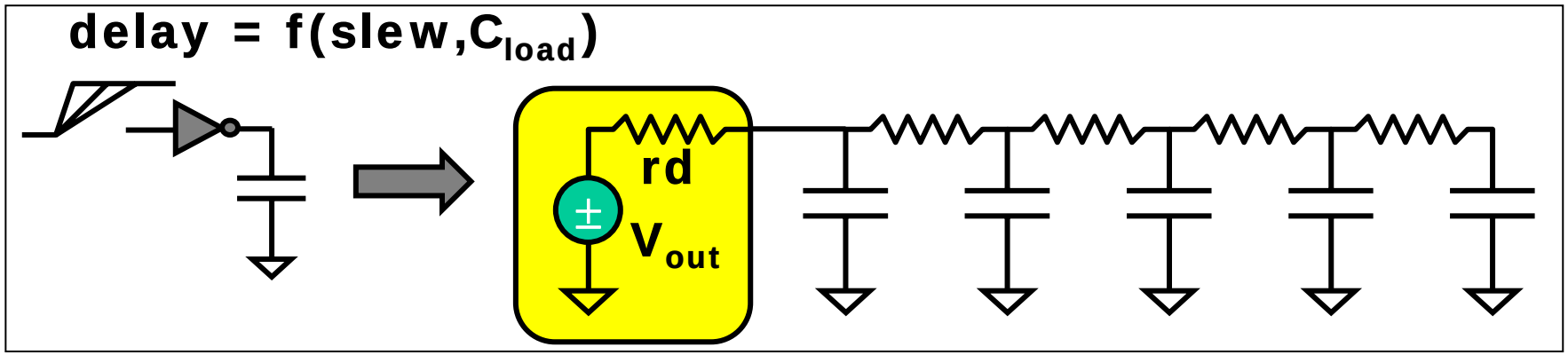
  - Gate driving total capacitance of net?
    - Gives erroneous results due to *resistive shielding*
  - Actual effective capacitance < total wiring capacitance
  - Techniques exist for determining $C_{effective}$, or modeling the gate using a voltage/current source

Computing C_{eff}: Overall flow

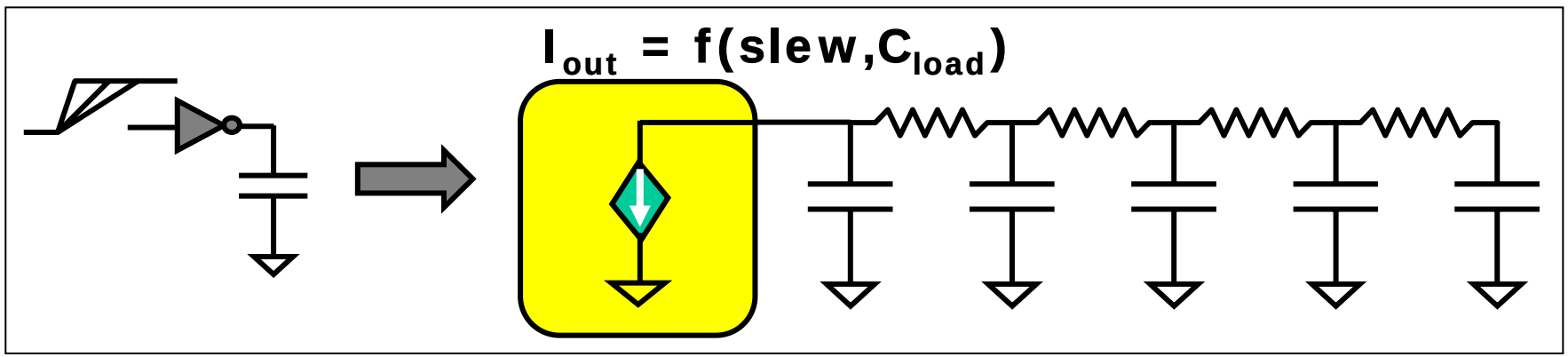


Current source model

- Represents the transistor I-V curve as a function of input slew and output load
- Linear Thevenin driver



- CCSM (Synopsys), ECSM (Cadence)

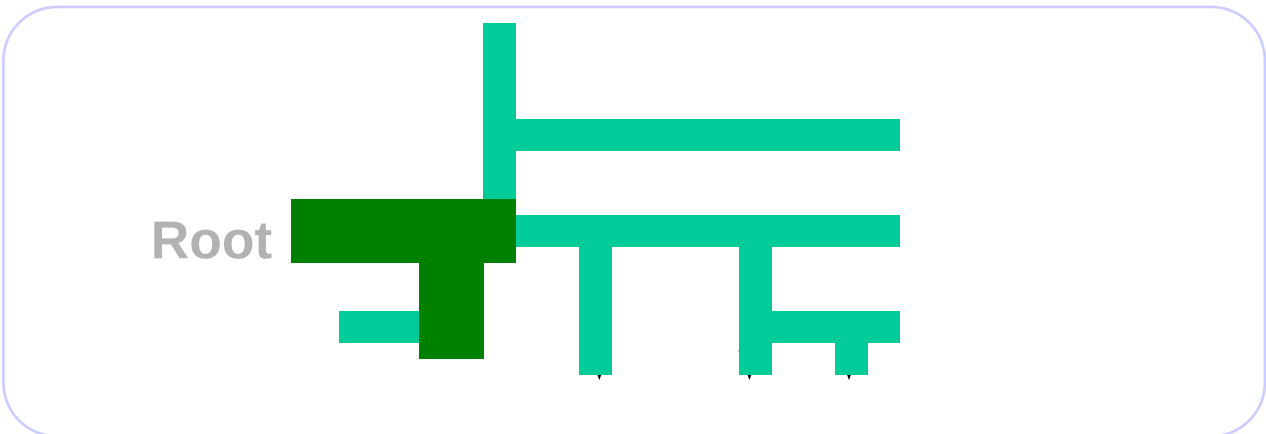


[Amin, DAC06]

Wire tapering and layer assignment

- **Elmore delay**

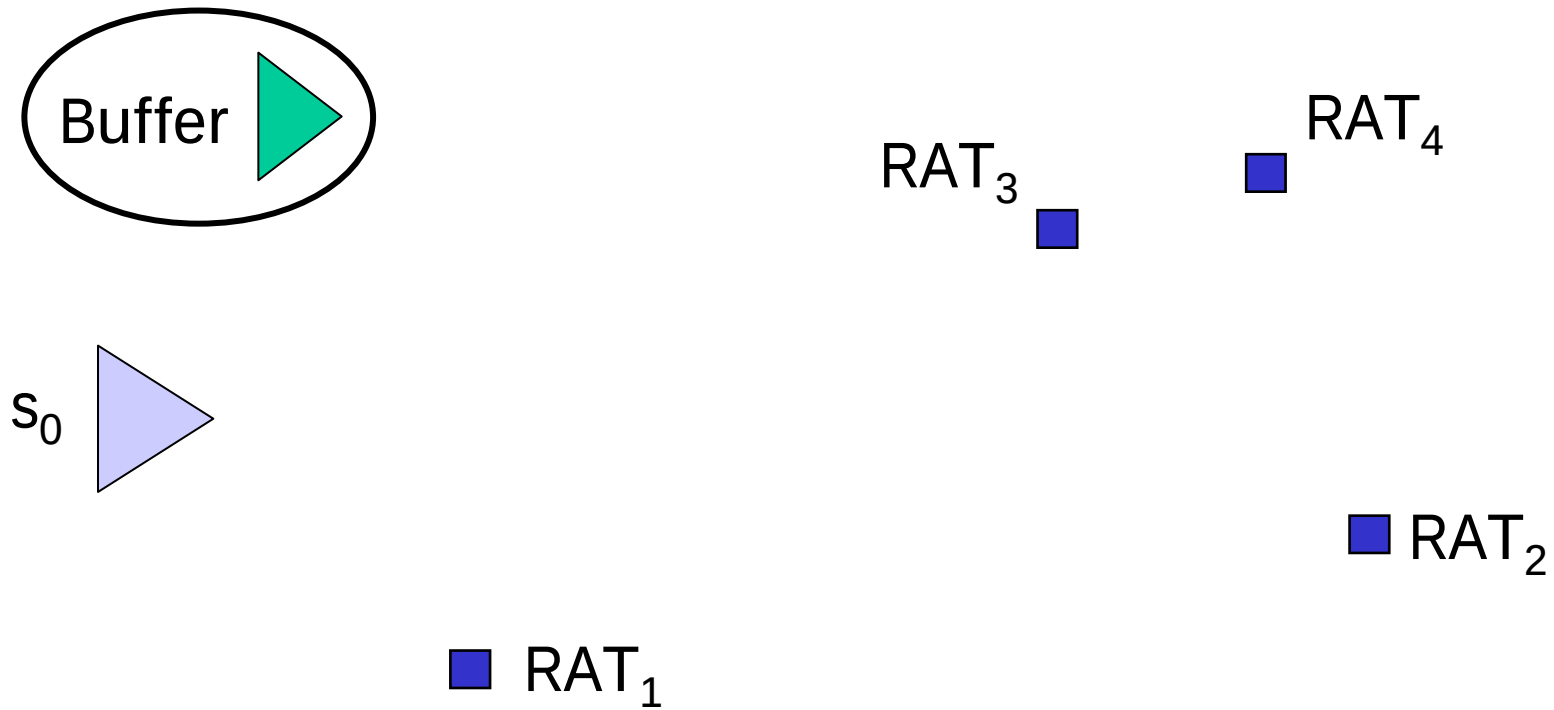
$$T_{D,k} = \sum_{i \in Path(k)} R_i \sum_{j \in downstream(i)} C_j$$



  - **Wires near the root must have low resistances**
  - **Wires near the leaves must have low capacitances**
  - **Wider wires near root, narrower near leaves**
    - **In practice: # of wire widths limited to two or three**
- **Same principle applies to layer assignment**

Simple buffer insertion problem

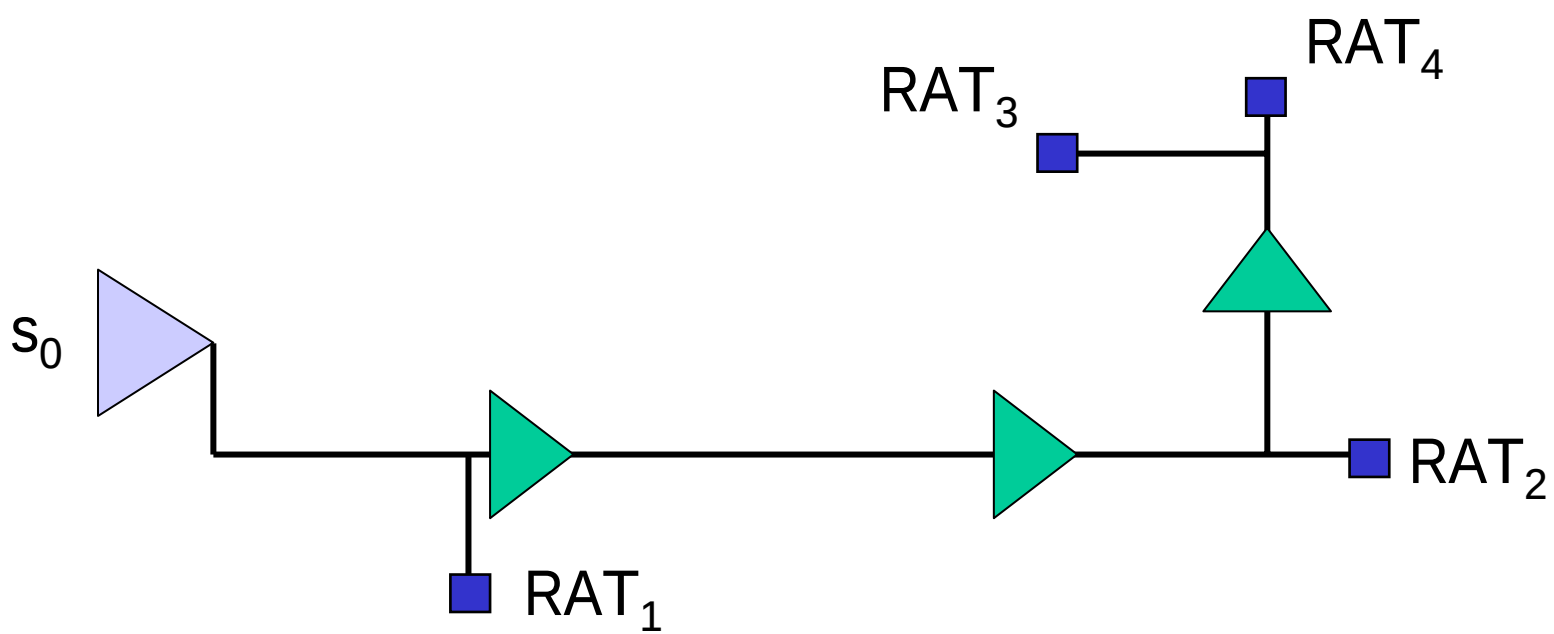
Given: Source and sink locations, sink capacitances and RATs, a buffer type, source delay rules, unit wire resistance and capacitance



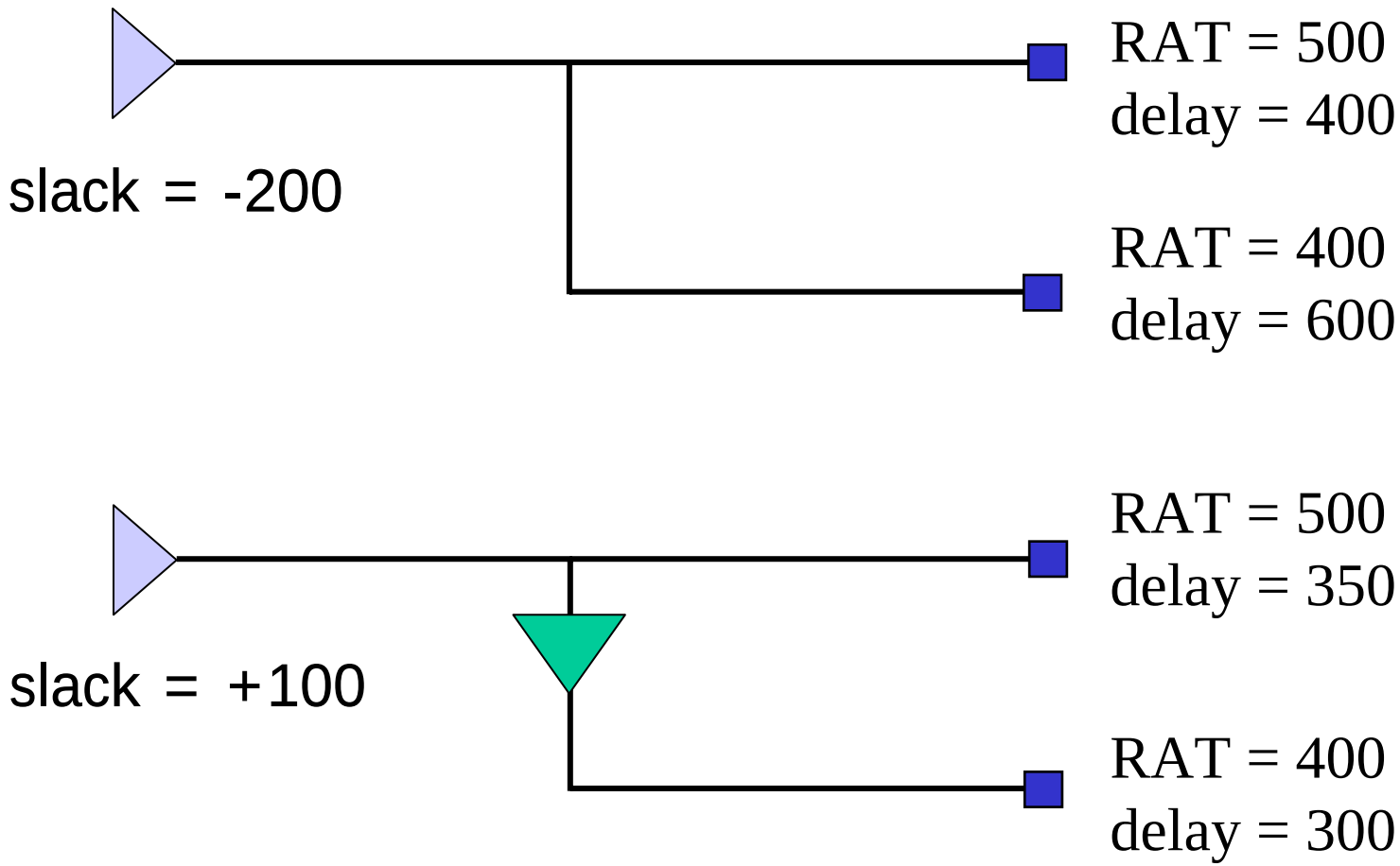
Simple buffer insertion problem

Find: Buffer locations and a routing tree such that slack at the source is minimized

$$q(s_0) = \min_{1 \le i \le 4} \{RAT(s_i) - delay(s_0, s_i)\}$$



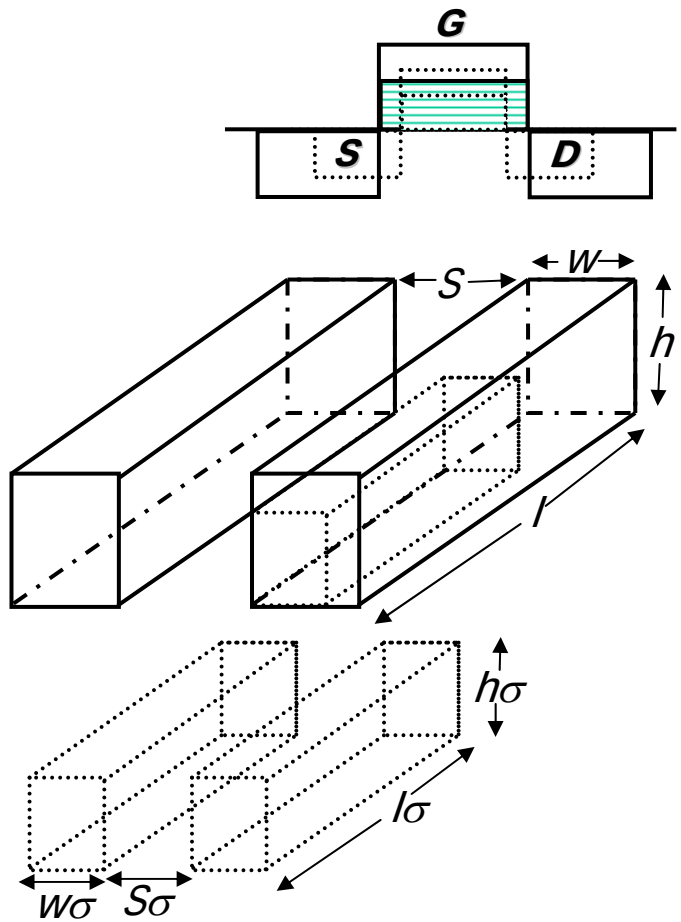
Slack example



Interconnects and Scaling Theory

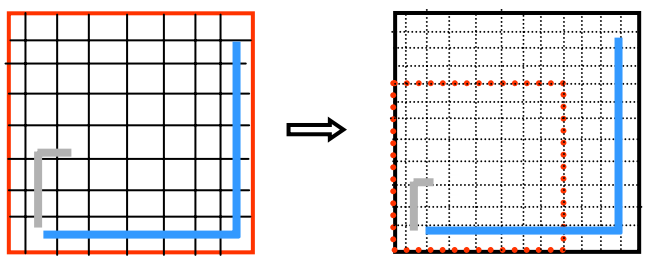
A scaling primer

- **Ideal process scaling:**
  - **Device geometries shrink by σ (= 0.7x)**
    - **Device delay shrinks by σ**
  - **Wire geometries shrink by σ**
    - ***Resistance*:** $\rho\, l/(w\sigma.h\sigma) = R/\sigma^2$
    - ***Coupling cap*:** $\varepsilon\,(h\sigma)\, l\,/(S\sigma)$ **= same**
    - ***Capacitance to ground*: *similar***
    - **In each process generation**
      ***R* doubles, *C* and *Cc* unchanged**
- **But it doesn't quite work that way**
  - ***h*** **scales by less than σ to control *R***



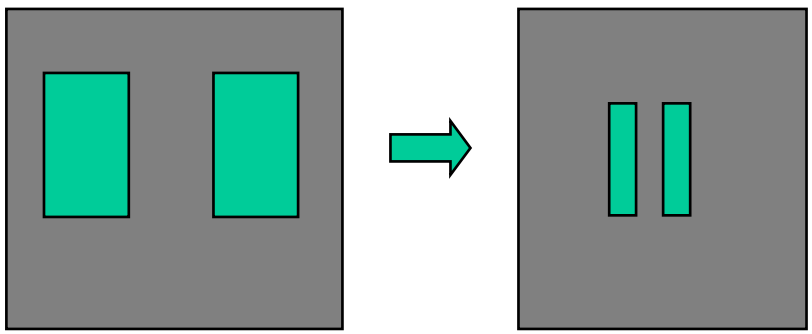
Block scaling

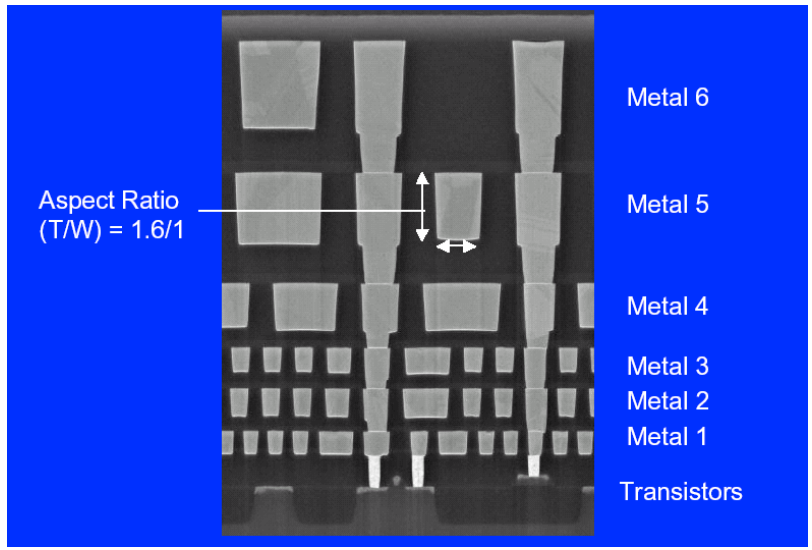
- **Block area often stays same**
  - **# cells, # nets doubles**
- **Wiring histogram shape (almost) invariant**
  - **Global interconnect lengths don't shrink**
  - **Local interconnect lengths shrink by σ**



A typical chip cross-section

- **Wires become "fatter" as you move to upper layers**
- **From one technology to the next, wire aspect ratios become more skewed**





[Intel]

- **R is controlled, at the expense of coupling capacitance**

The role of interconnects

- **Short interconnect**
  - Used to connect nearby cells, $R_{driver} >> R_{interconnect}$
  - Minimize wire C, i.e., use short minwidth wires
- **Medium to long-distance ("global") interconnect**
  - $R_{driver} \approx R_{interconnect}$
  - Size wires to tradeoff area vs. delay
  - Increasing width $\Rightarrow$ Capacitance increases, Resistance decreases
- **"Fat" wires**
  - Thicker cross-sections in higher metal layers
  - Useful for reducing delays for global wires
  - Inductance issues, sharing of limited resource

Interconnect delay scaling

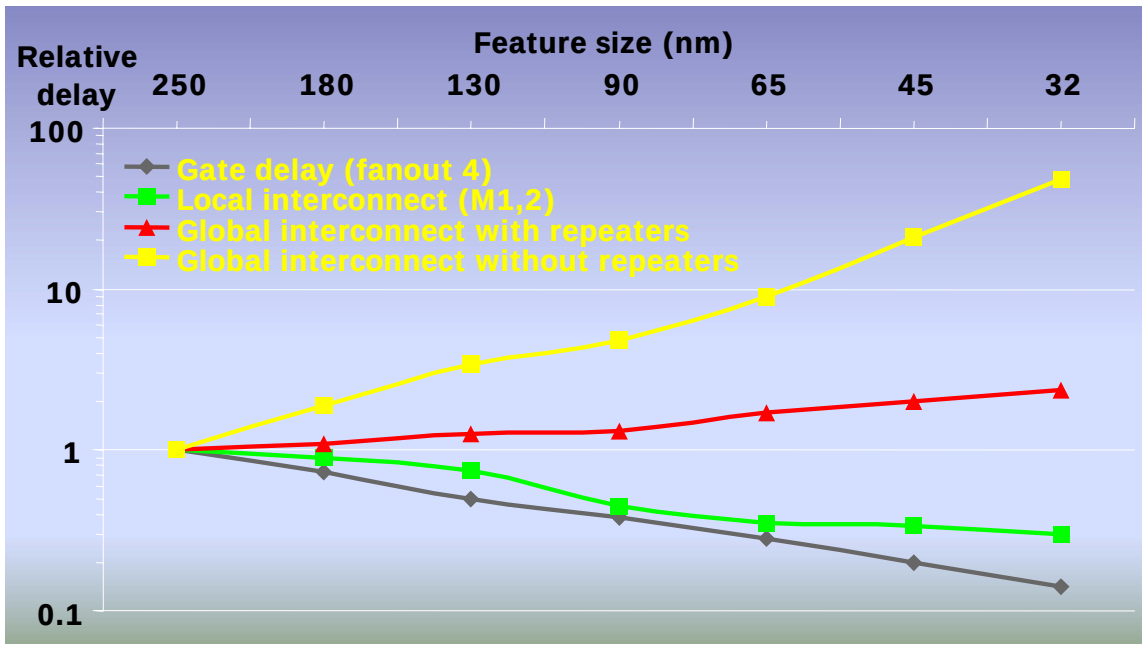
- **Delay of a wire of length *l* :**
  $\tau_{int} = (rl)(cl) = rcl^2$ **(first order)**

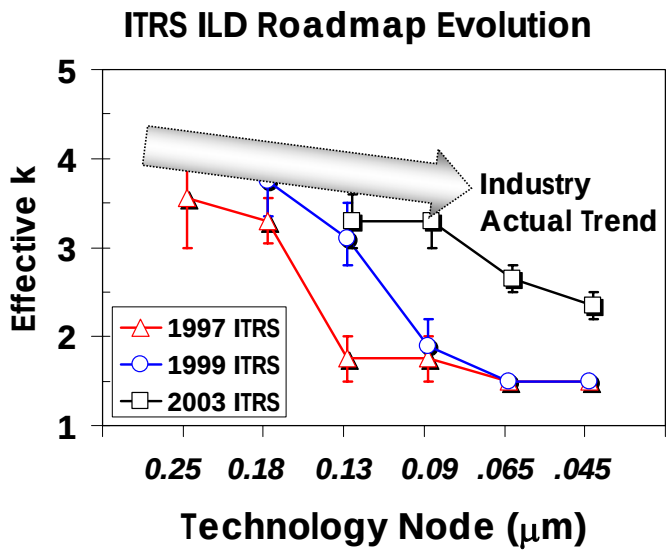
- **Local interconnects :**
  $\tau_{int} : (r/\sigma^2)(c)(l\sigma)^2 = rcl^2$
  - **Local interconnect delay unchanged (but devices get faster)**

- **Global interconnects :**
  $\tau_{int} : (r/\sigma^2)(c)(l)^2 = (rcl^2)/\sigma^2$
  - **Global interconnect delay doubles – unsustainable!**
  - **Problem somewhat mitigated using buffers, using nonideal scaling as outlined earlier**

- ***Interconnect delay increasingly more dominant***

ITRS projections



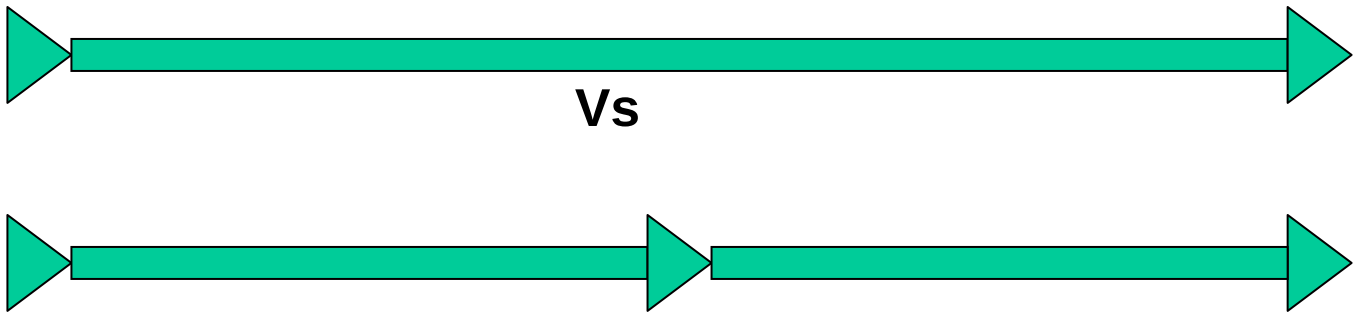


Source: Chia Hong Jan, IEDM 2003 Interconnect Short Course

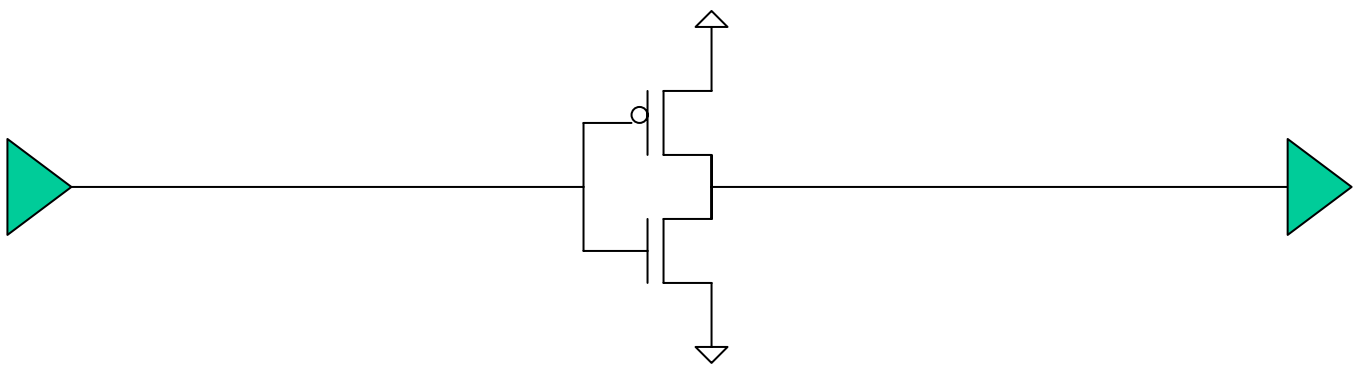
ITRS projections often a “best case scenario” projection

Buffer insertion

- **Consider**

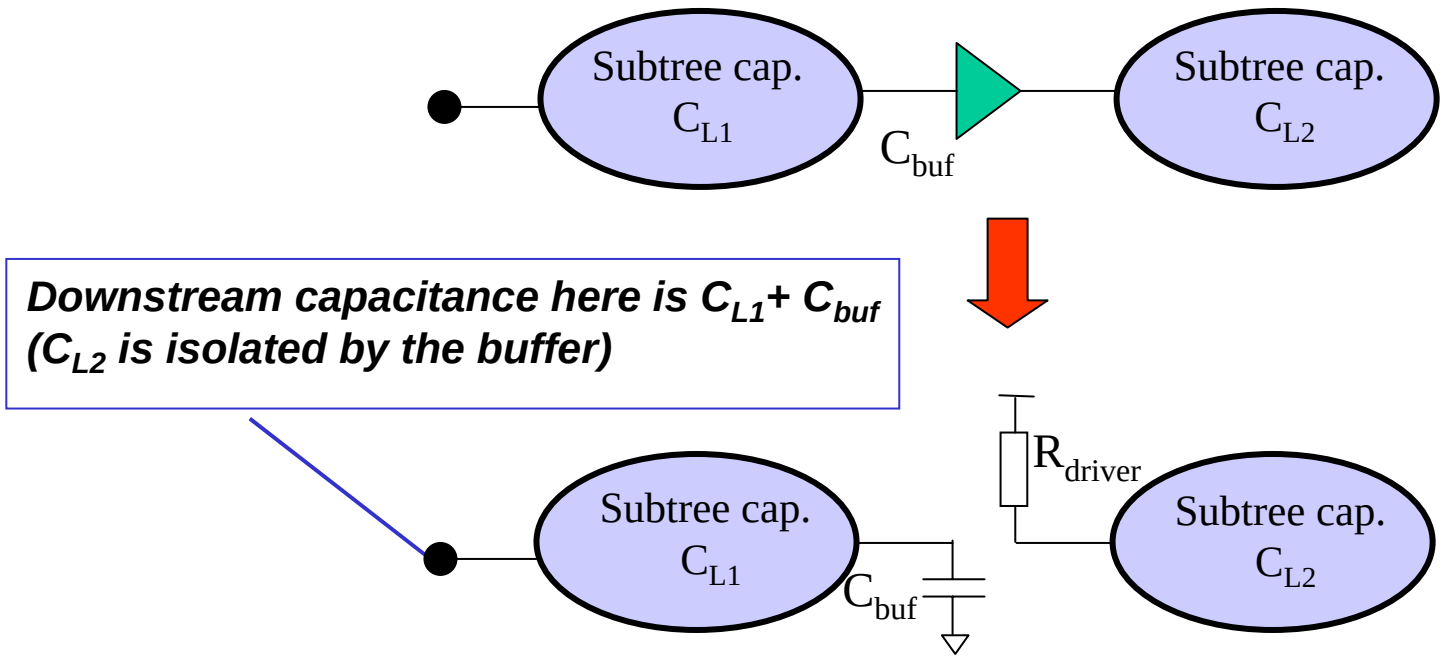


- **A buffer effectively isolates the downstream capacitance**

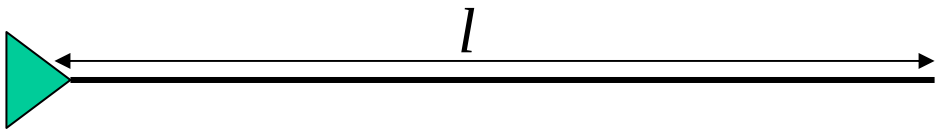


Optimizing medium/long interconnects

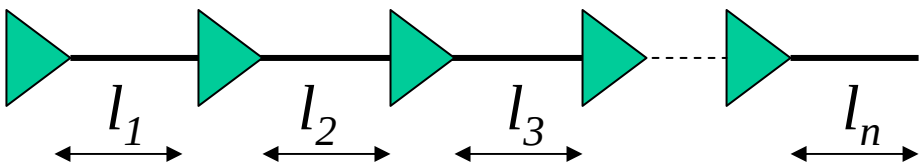
- **Delays of interconnects may become very large**
- **Wire sizing helps to control the delay**
- ***Repeater insertion* is another effective technique**
- **Effects of a buffer**
  - **Isolates load capacitances of different "stages"**
  - **Adds a delay**



Buffered global interconnects: Intuition



Interconnect delay = $r.c.l^2$



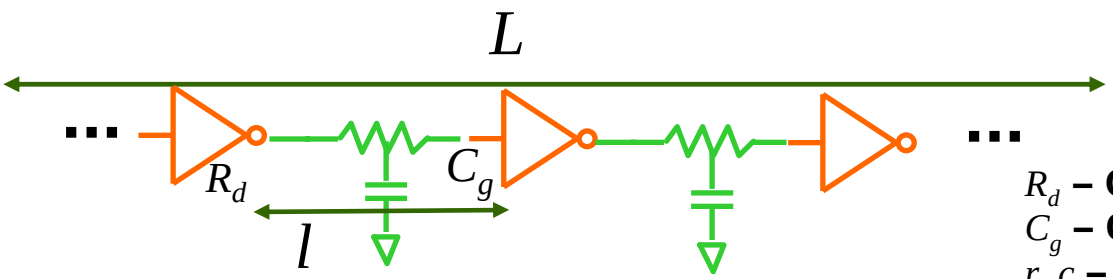
Now, interconnect delay = $\Sigma\, r.c.l_i^2 < r.c.l^2$ *(where $l = \Sigma\, l_j$)*

since $\Sigma\,(l_j^{\,2}) < (\Sigma\, l_j)^2$

(Of course, account for intrinsic buffer delay also)

More precise analysis: Optimal inter-buffer length

- **First order (lumped parasitic, Elmore delay) analysis**



R_d – **On resistance of inverter**
C_g – **Gate input capacitance**
r, c – **Resistance, cap. per micron**

- **Assume N identical buffers with equal inter-buffer length l ($L = Nl$)**

$$T = N\left[R_d\left(C_g + cl\right) + rl\left(C_g + cl\right)\right]$$

$$= L\left[rcl + \left(rC_g + R_d c\right) + \frac{1}{l}\left(R_d C_g\right)\right]$$

- **For minimum delay,**

$$\frac{dT}{dl} = 0 \quad \Rightarrow \quad L\left[rc - \frac{R_d C_g}{l_{opt}^2}\right] = 0 \quad \Rightarrow \quad \boxed{l_{opt} = \sqrt{\frac{R_d C_g}{rc}}}$$

Optimal interconnect delay

- **Substituting l_{opt} back into the interconnect delay expression:**

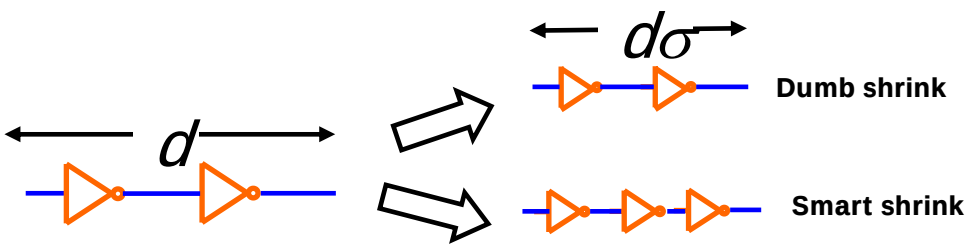
$$T_{opt} = L\left[rcl_{opt} + \left(rC_g + R_d c\right) + \frac{1}{l_{opt}}\left(R_d C_g\right)\right]$$

$$T_{opt} = L\left[2\sqrt{R_d C_g rc} + \left(rC_g + R_d c\right)\right]$$

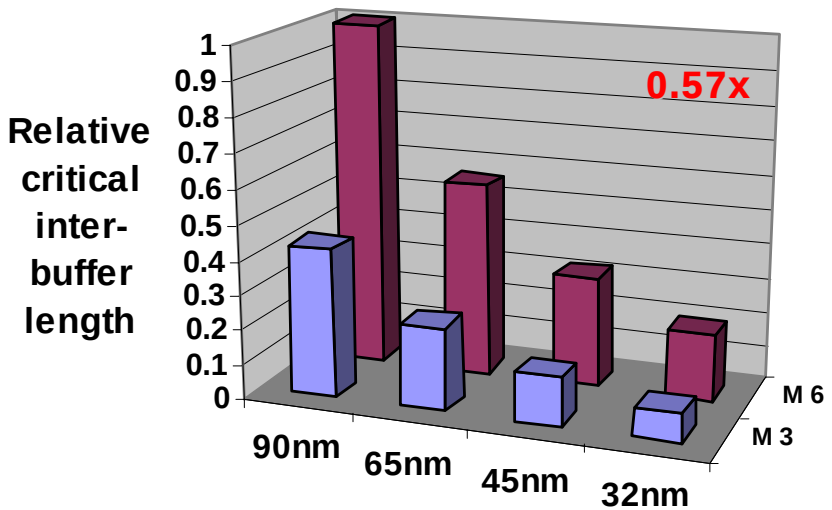
Delay grows linearly with L (instead of quadratically)

$$l_{opt} = \sqrt{\frac{R_d C_g}{rc}}$$

Buffer-to-buffer spacing reduces in successive technology nodes



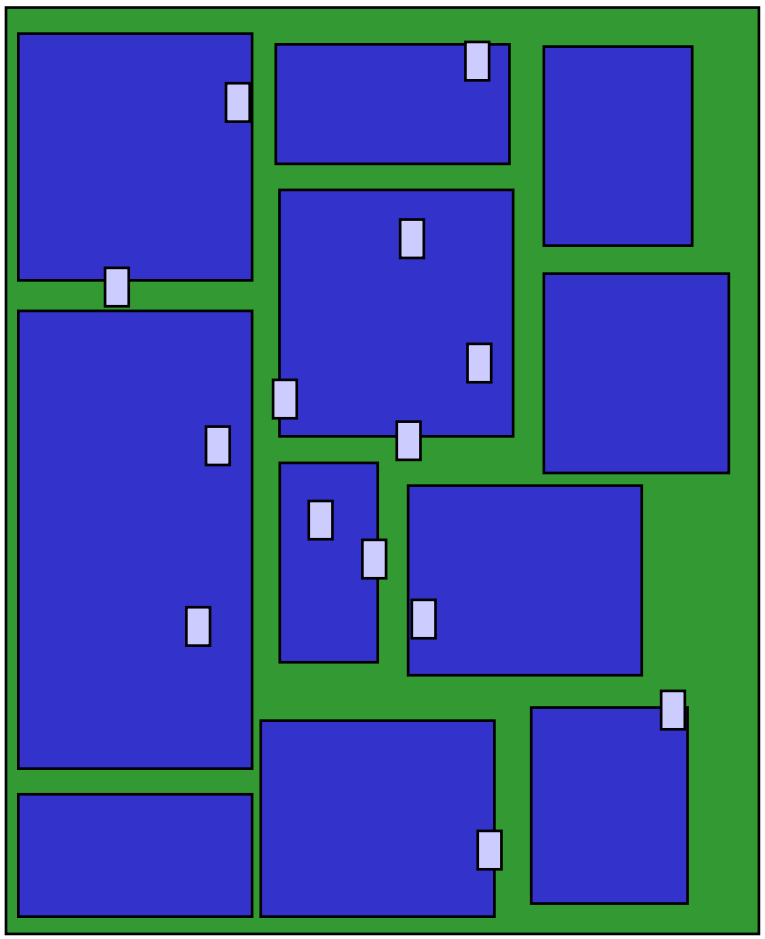
Critical inter-buffer lengths



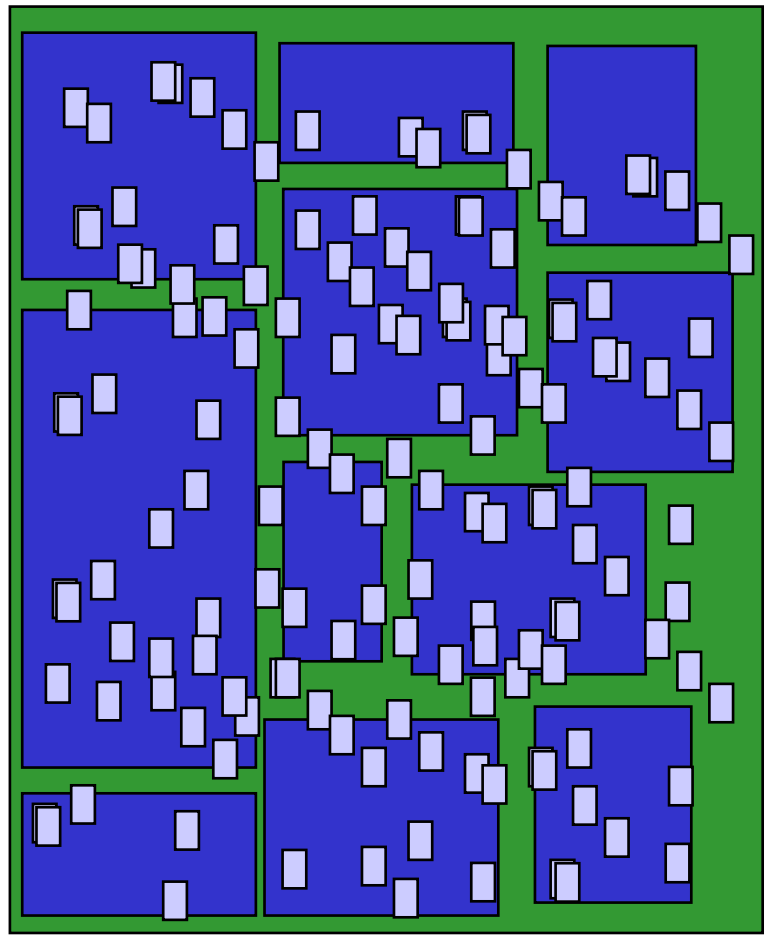
In line with scaling theory: $\sigma\sqrt{\sigma} = 0.586$

- **Study based on exhaustive SPICE simulation and projected process files (Saxena et al. TCAD'04)**
- **Optimally-sized uniformly for min delay**
  - ***Min distance at which inserting a buffer speeds up the line***
- ***"Ideally shrunk" circuit requires additional buffers***
  (0.7x *vs* 0.57x)

Buffer planning needed!

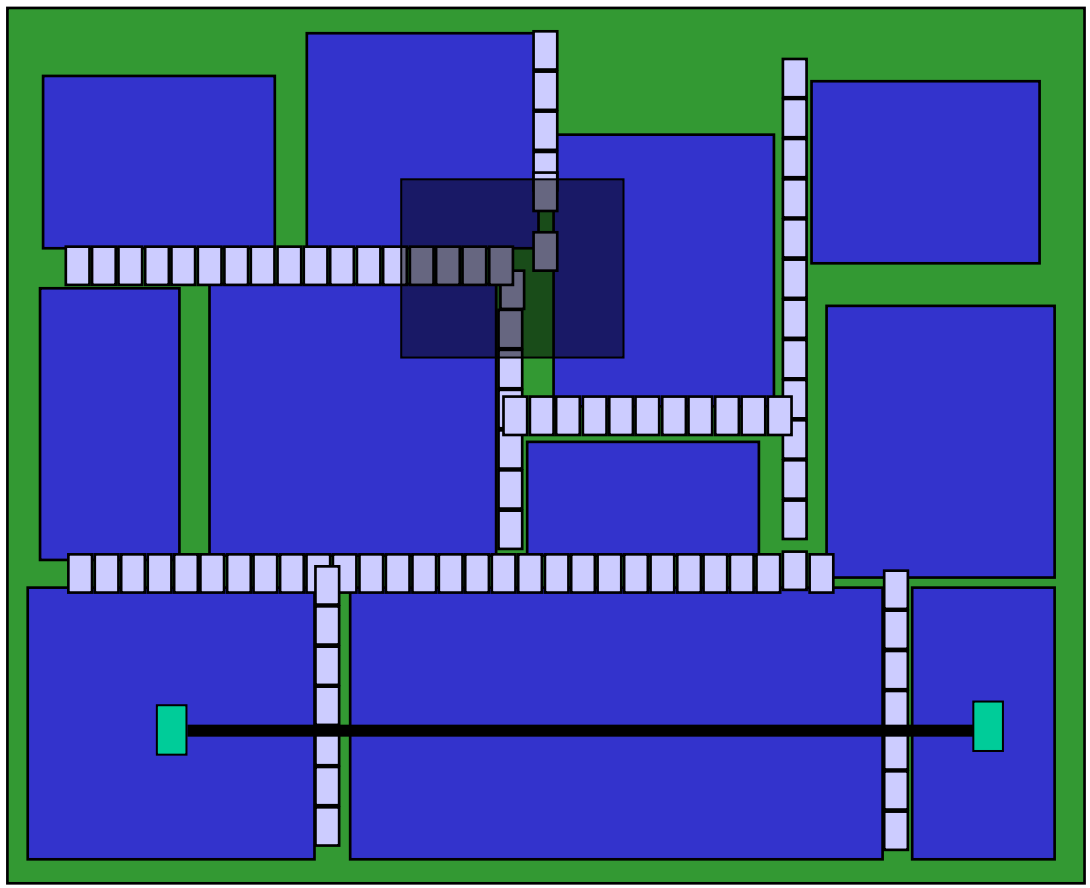


Past

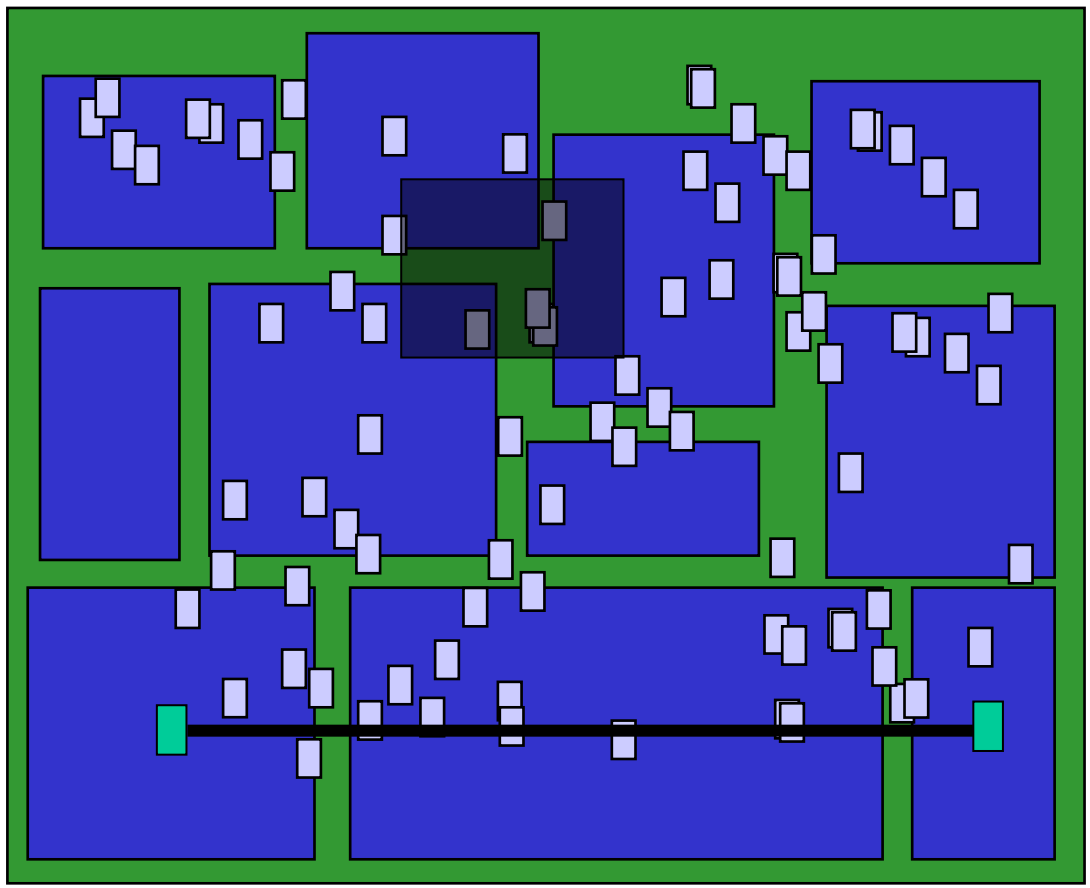


Present/Future

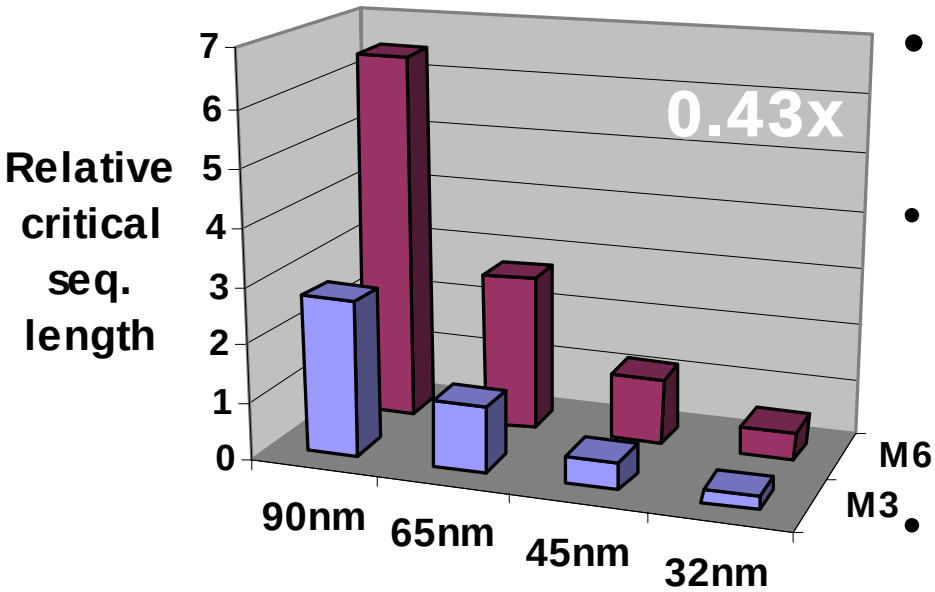
Buffer block planning



Buffer block planning



Critical sequential lengths

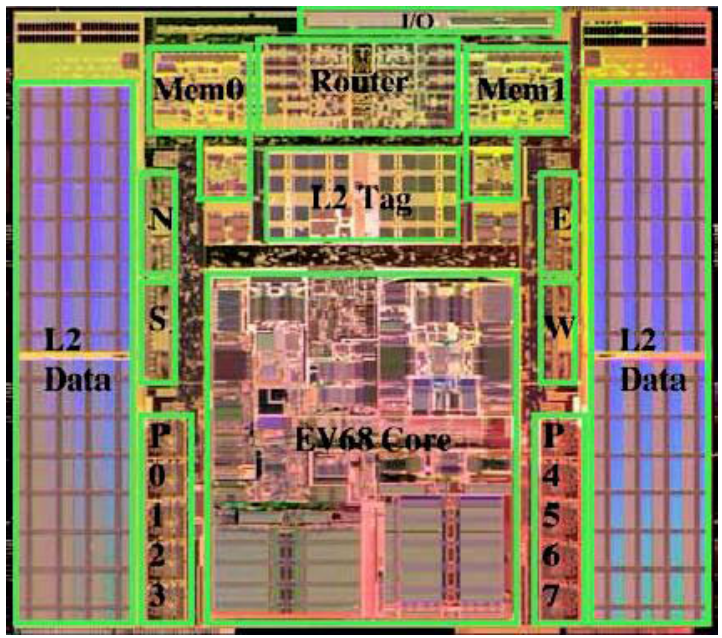


- **Optimized for max distance in one clock period**
- **Assumes:**
  - **2x frequency scaling**
  - ***Ignores setup, hold, skew***
- **Even with 1.4x ("Moore") frequency scaling, critical seq. lengths shrink at ~0.62x**
- ***"Ideally shrunk" circuit requires <u>much</u> new wire pipelining***

(0.7x *vs* 0.43x / 0.62x)

Architectural impact

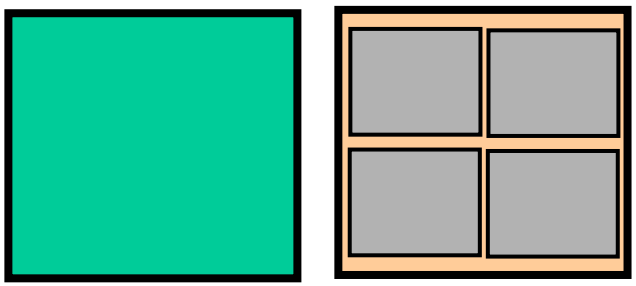
- **Example processor floorplan shown below**

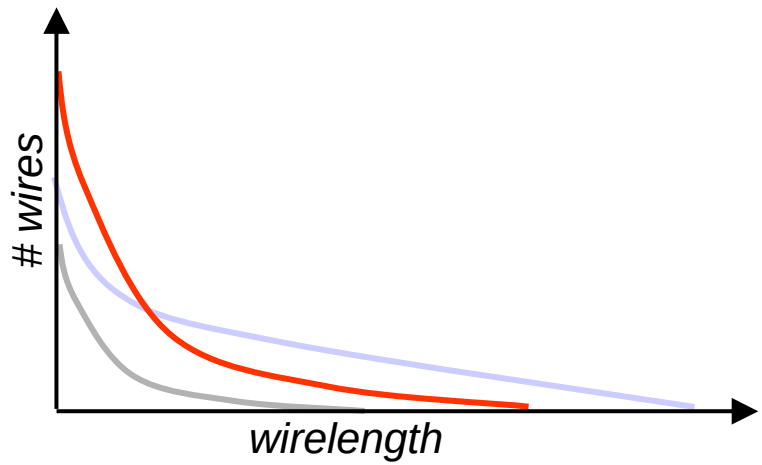


- **Layout decisions affect # clock cycles required to convey a signal**
  - **Architectural decisions must be made hand-in-hand with layout**

Longer term solution: architectural changes

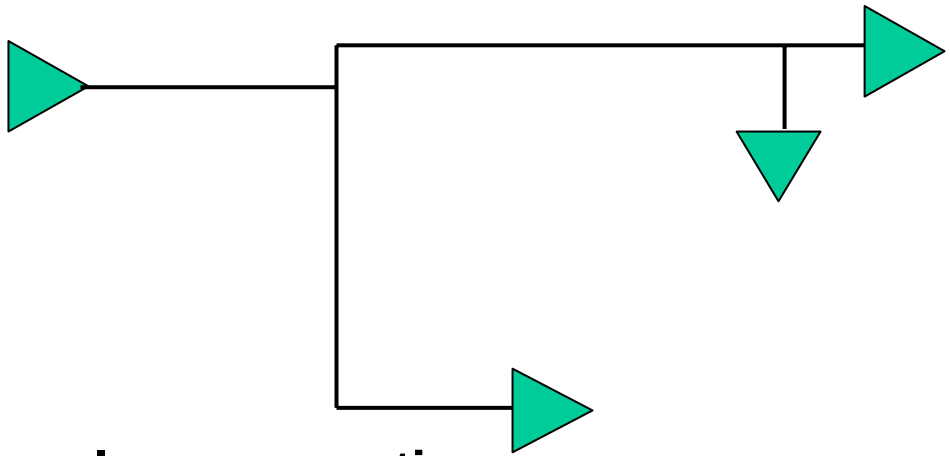
- **Simplify interconnection complexity architecturally**
  - **Modify wiring histogram shape (i.e. Rent's parameters) of design**
- **An example: multi-core microprocessors**
  - **Goes counter to traditional approach of increased integration through block size scaling**





Synthesis of Signal Interconnects

Signal interconnect synthesis



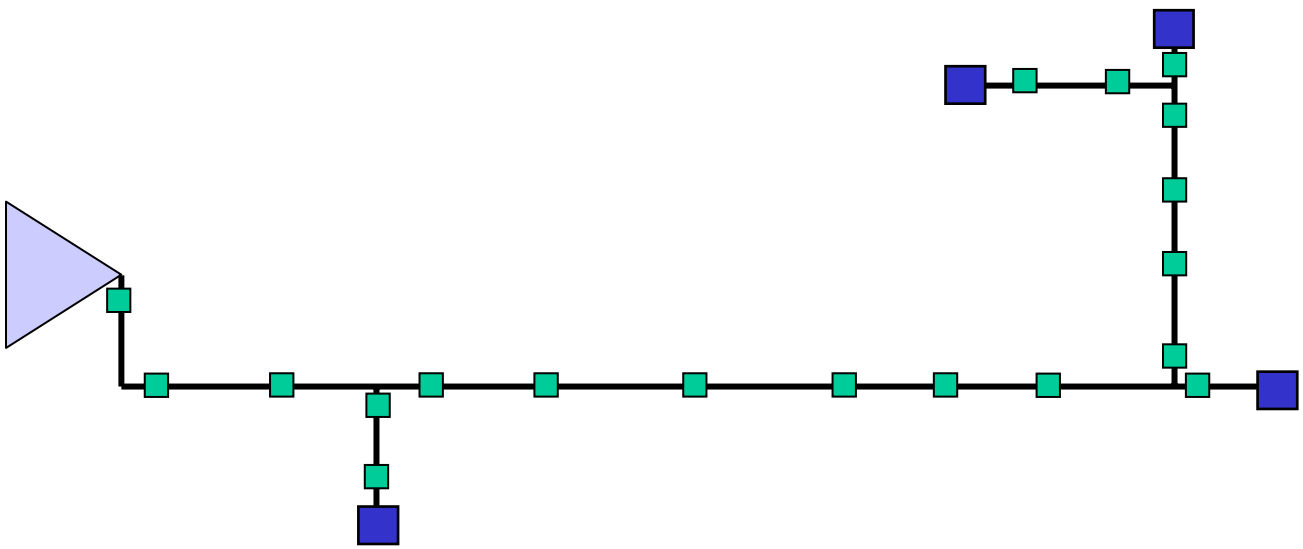
- Interconnect topology generation
- Interconnect delay optimization
- Noise optimization
- Bus design
- Congestion considerations

Van Ginneken's classic algorithm

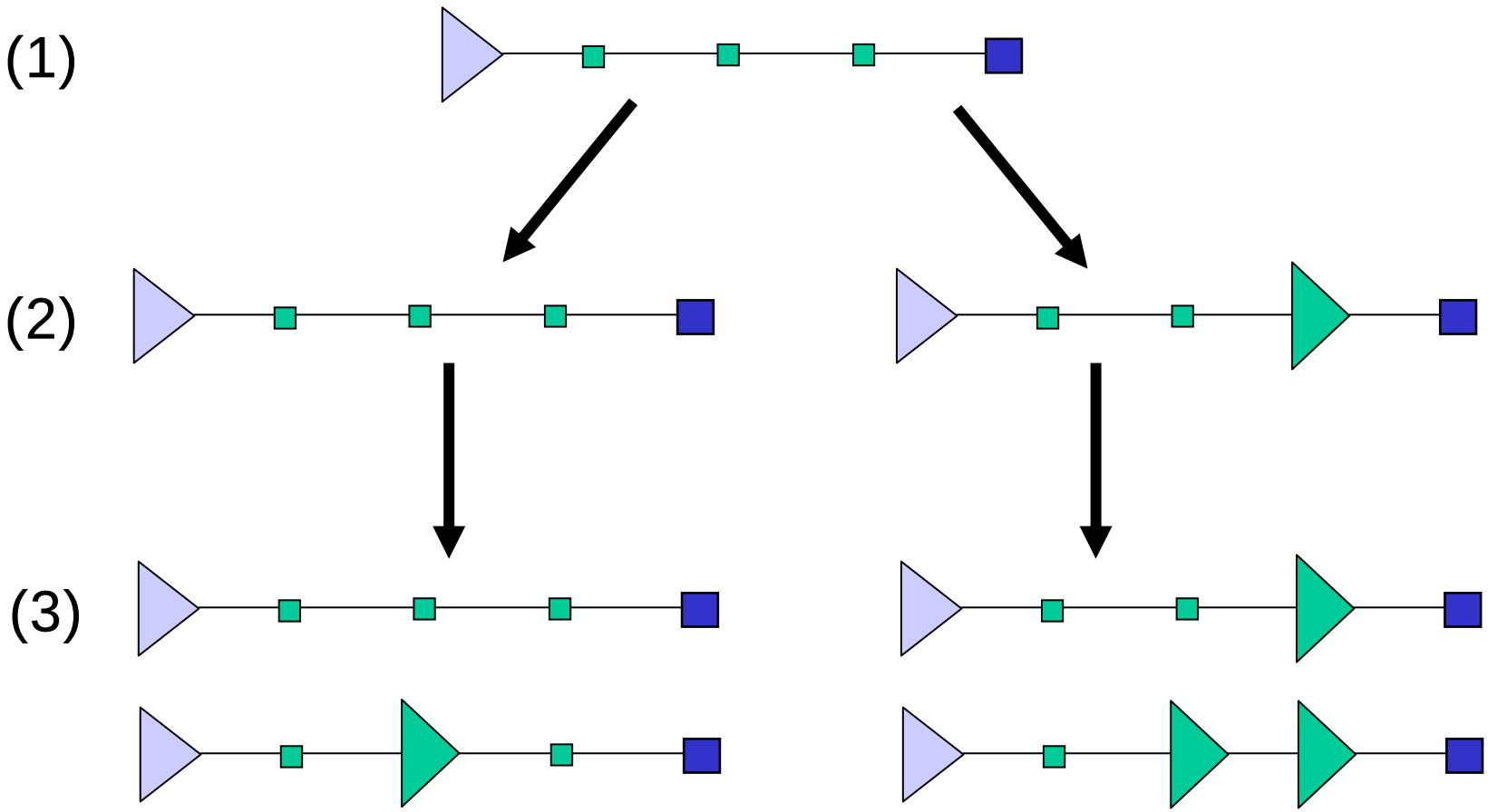
- Optimal for multi-sink nets
- Quadratic runtime
- Bottom-up from sinks to source
- Generate list of candidates at each node
- At source, pick the best candidate in list

Key assumptions

- Given routing tree
- Given potential insertion points

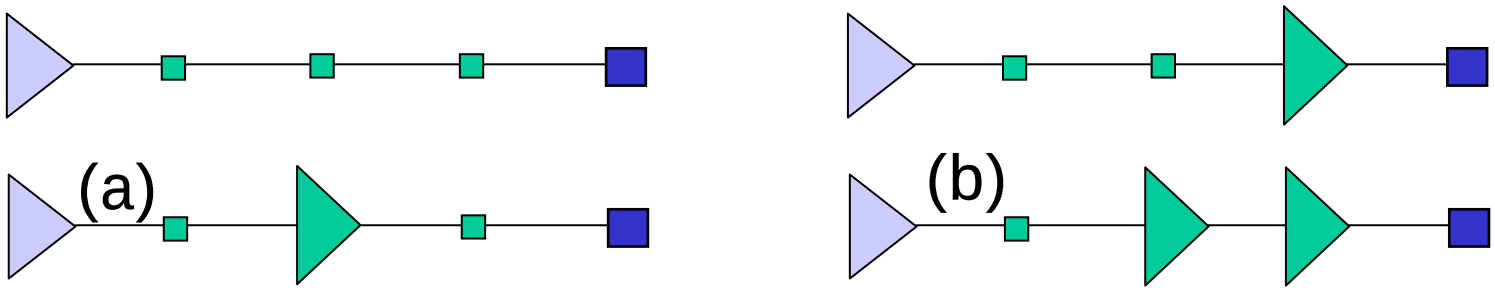


Generating candidates



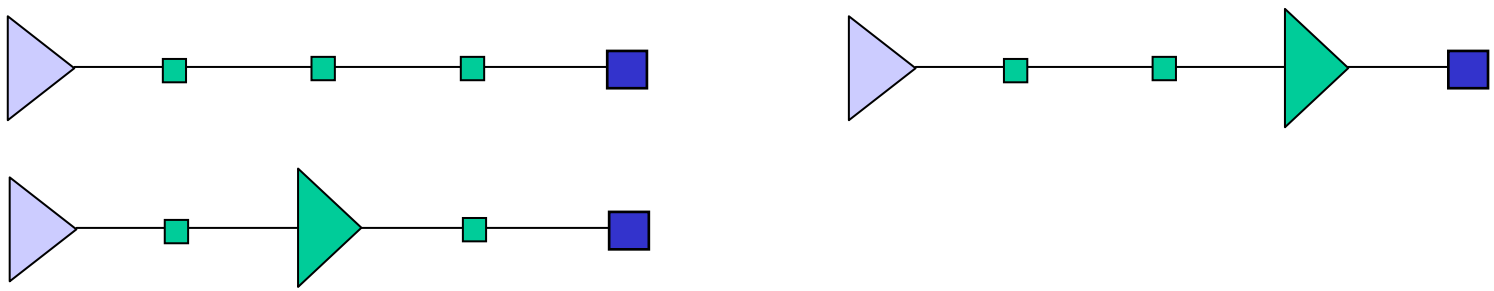
Pruning candidates

(3)

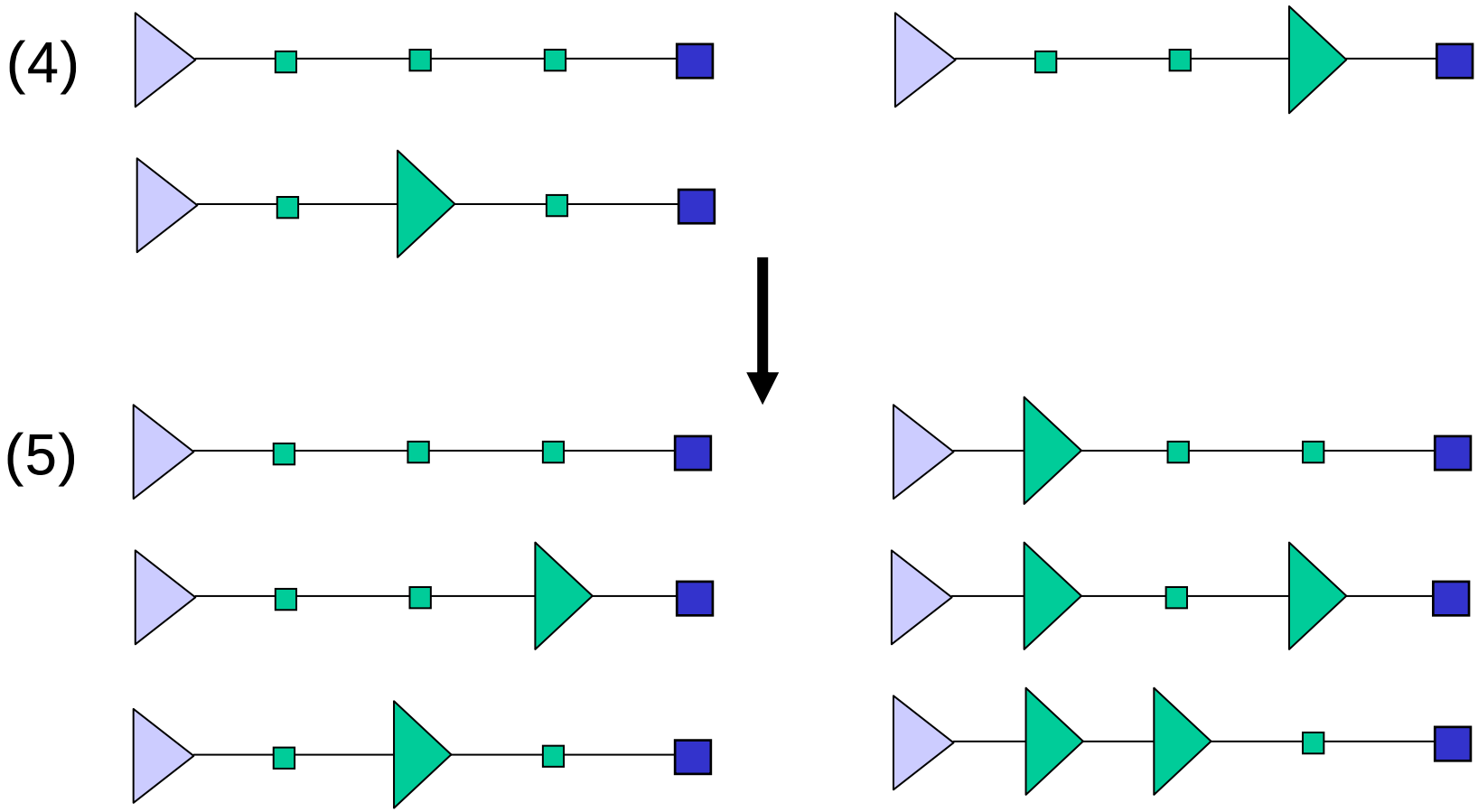


Both (a) and (b) “look” the same to the source.
Throw out the one with the worst slack

(4)



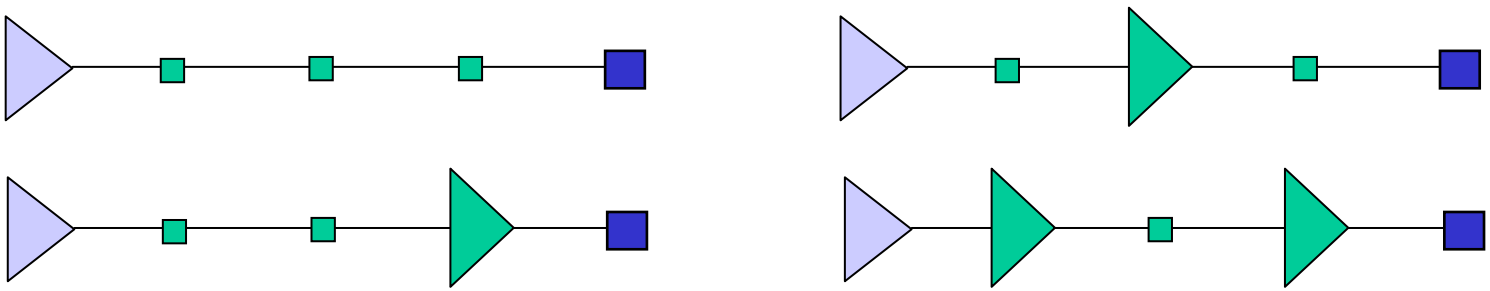
Candidate example (continued)



Candidate example (continued)

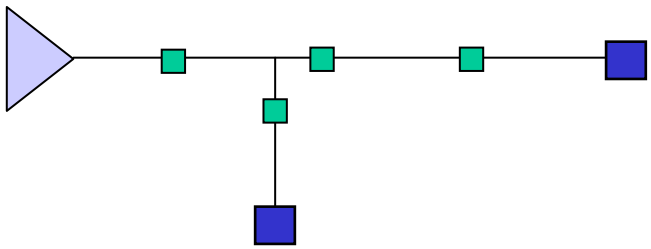
After pruning

(5)

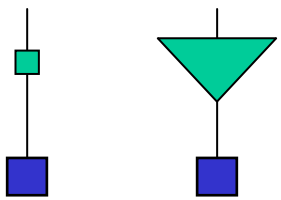


At driver, compute which candidate maximizes slack. Result is optimal.

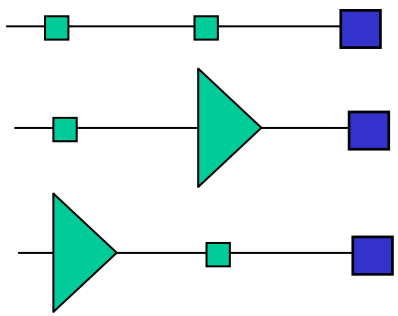
Merging branches

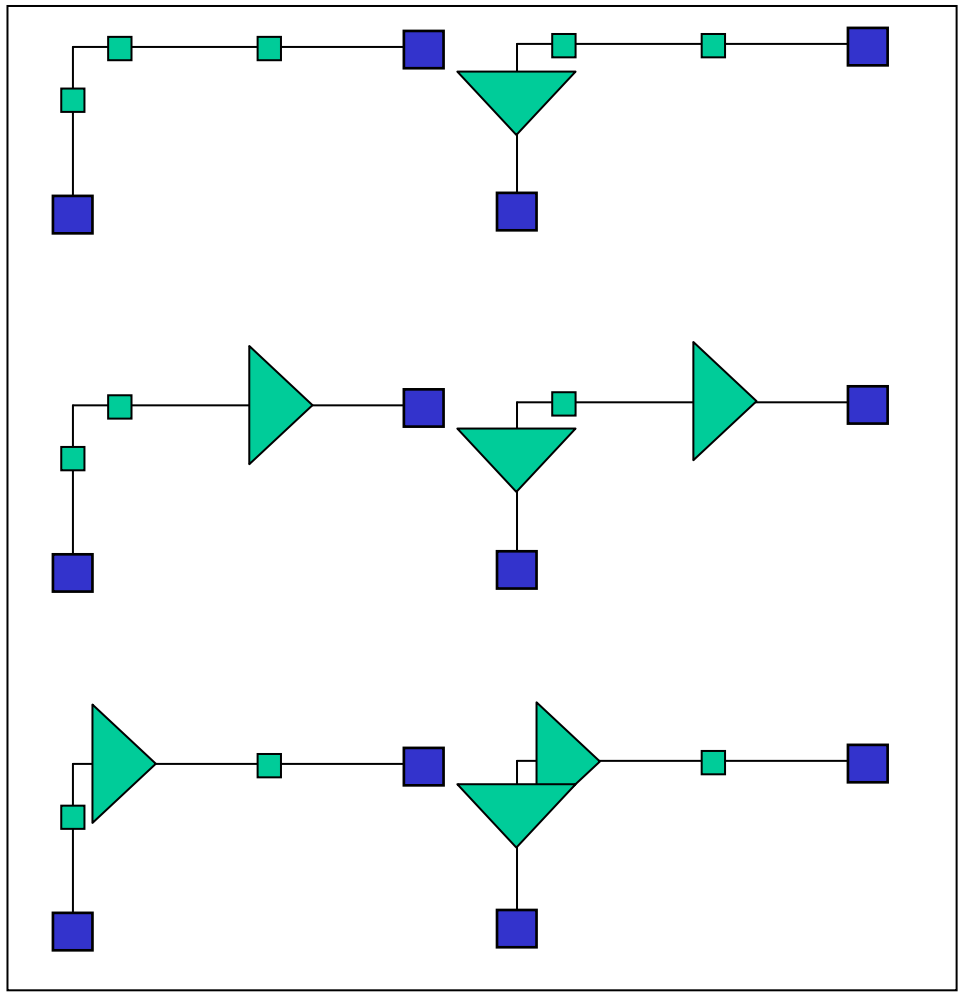


Left Candidates

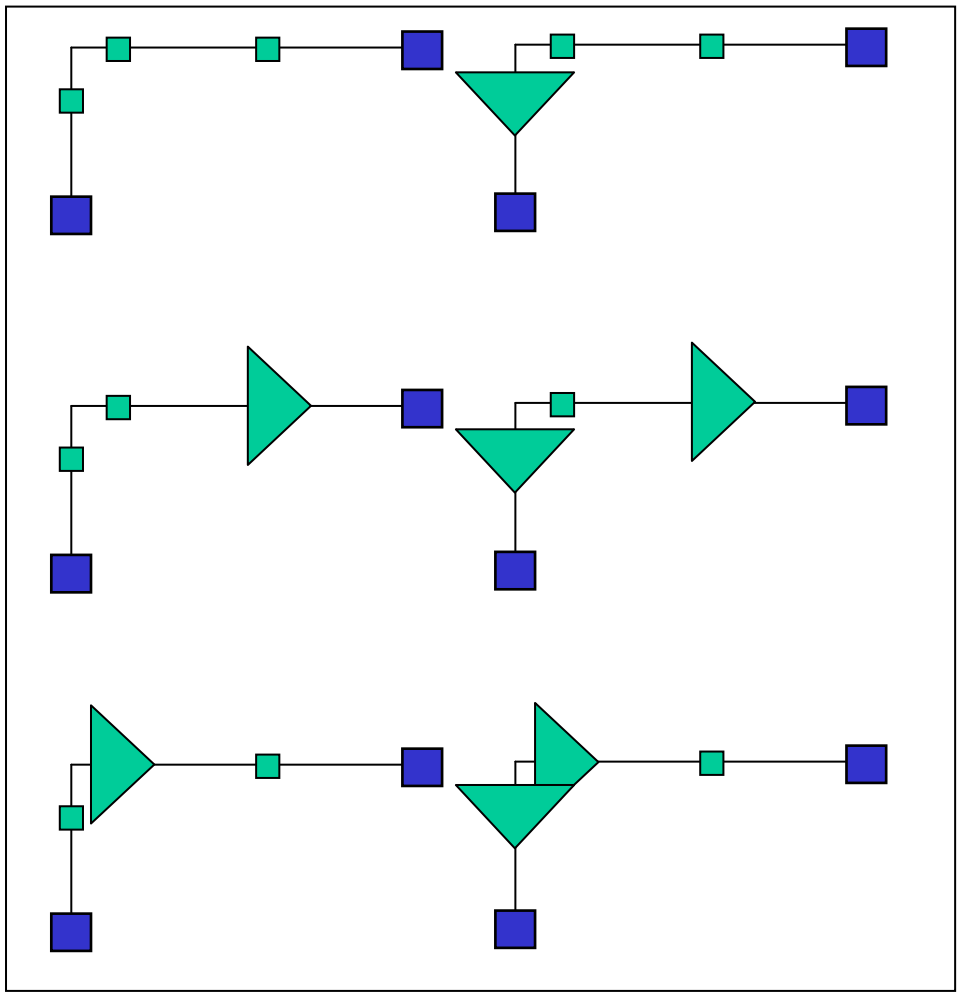


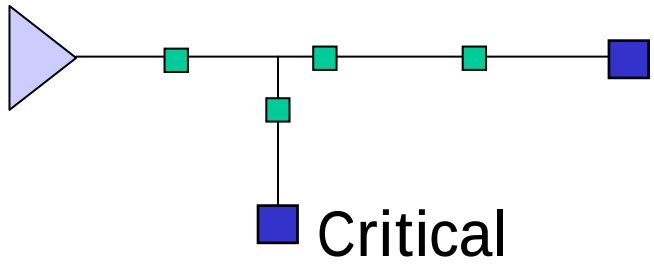
Right Candidates

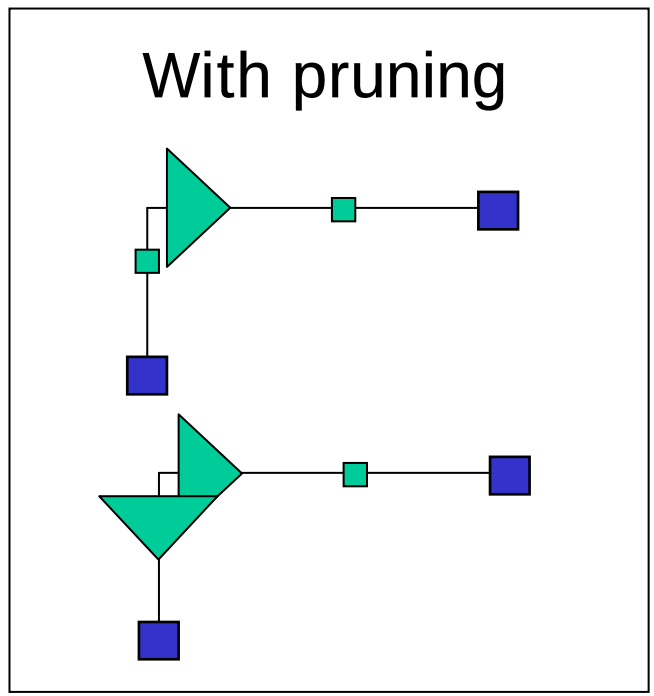




Pruning merged branches

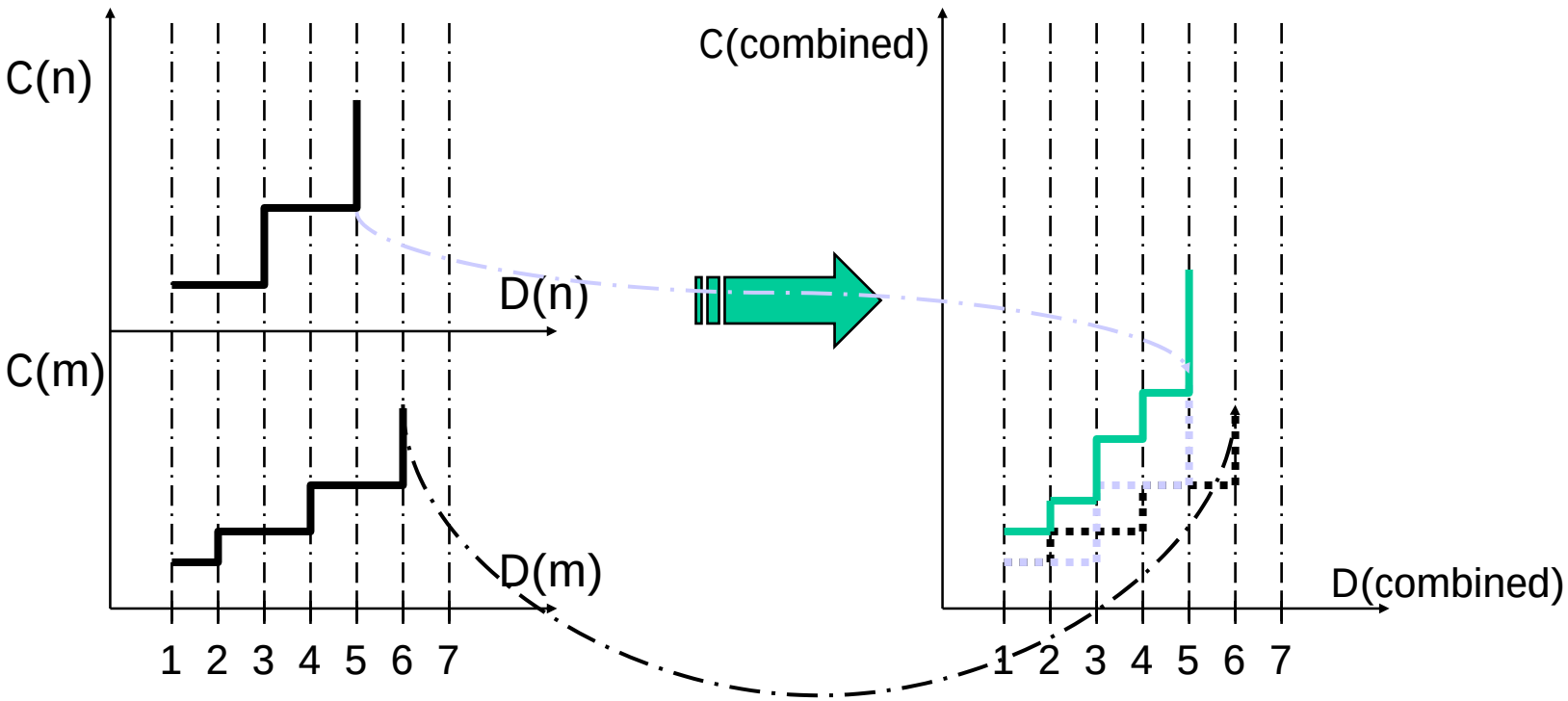




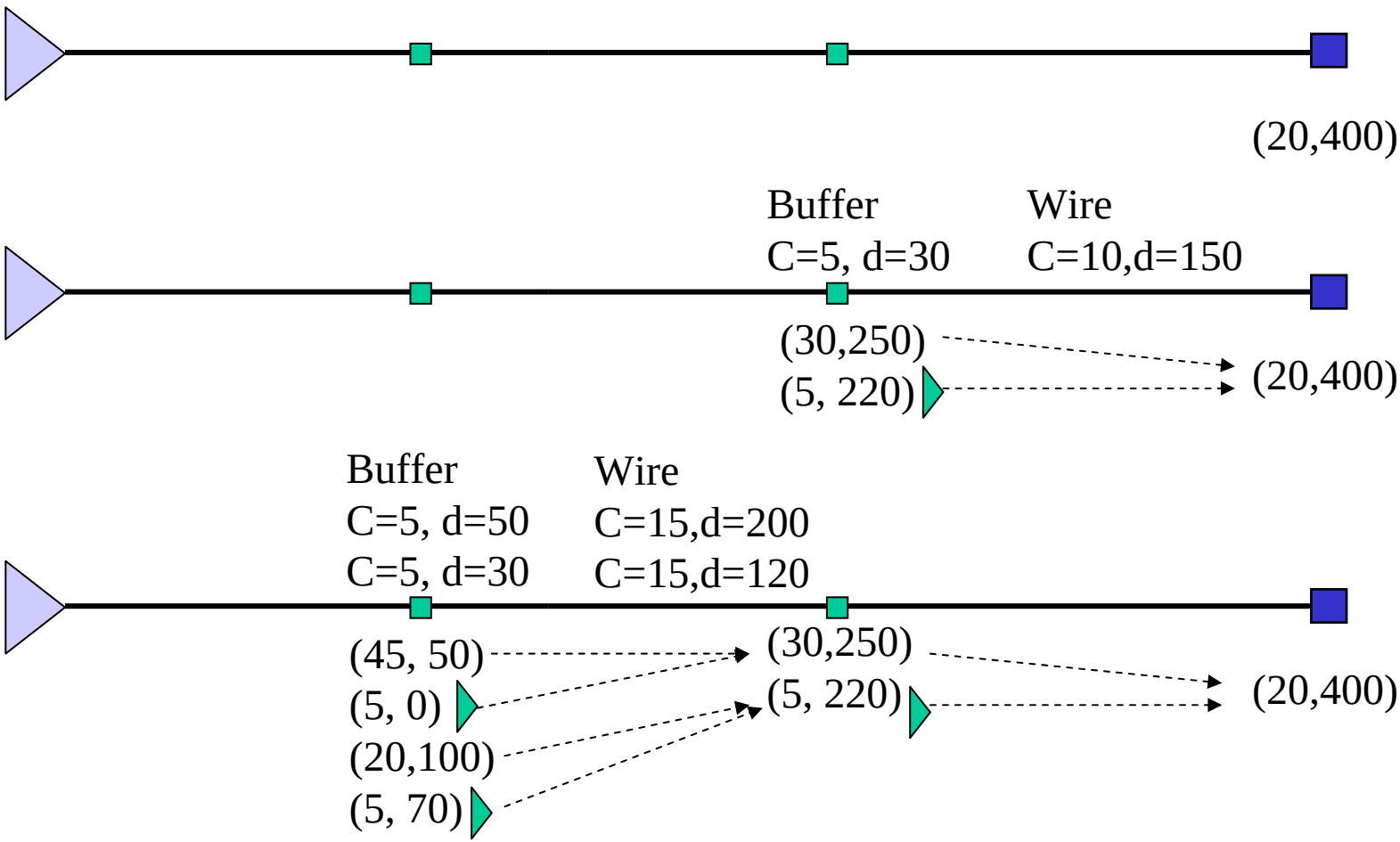


Combining the Options

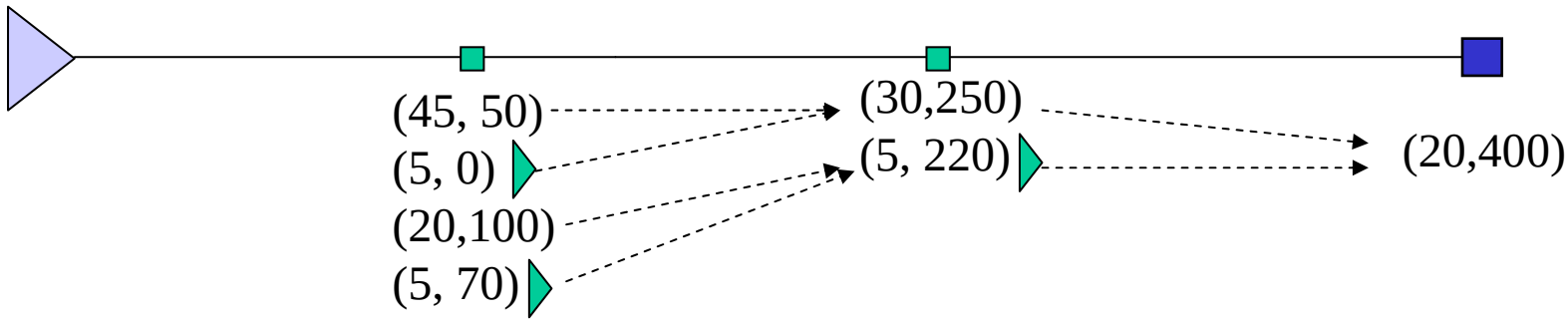
- **Draw a plot of all (C_k, D_k) pairs for both children m and n (assuming a binary tree)**



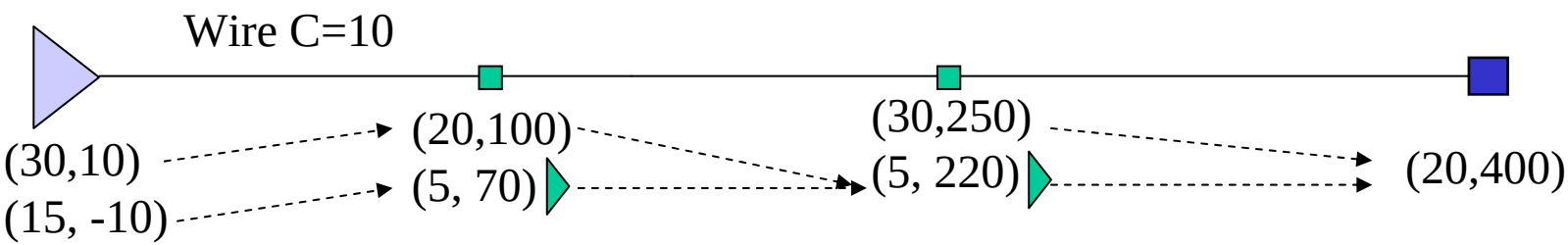
Van Ginneken example



Van Ginneken example (continued)



(5,0) is inferior to (5,70). (45,50) is inferior to (20,100)



Pick solution with largest slack, follow arrows to get solution

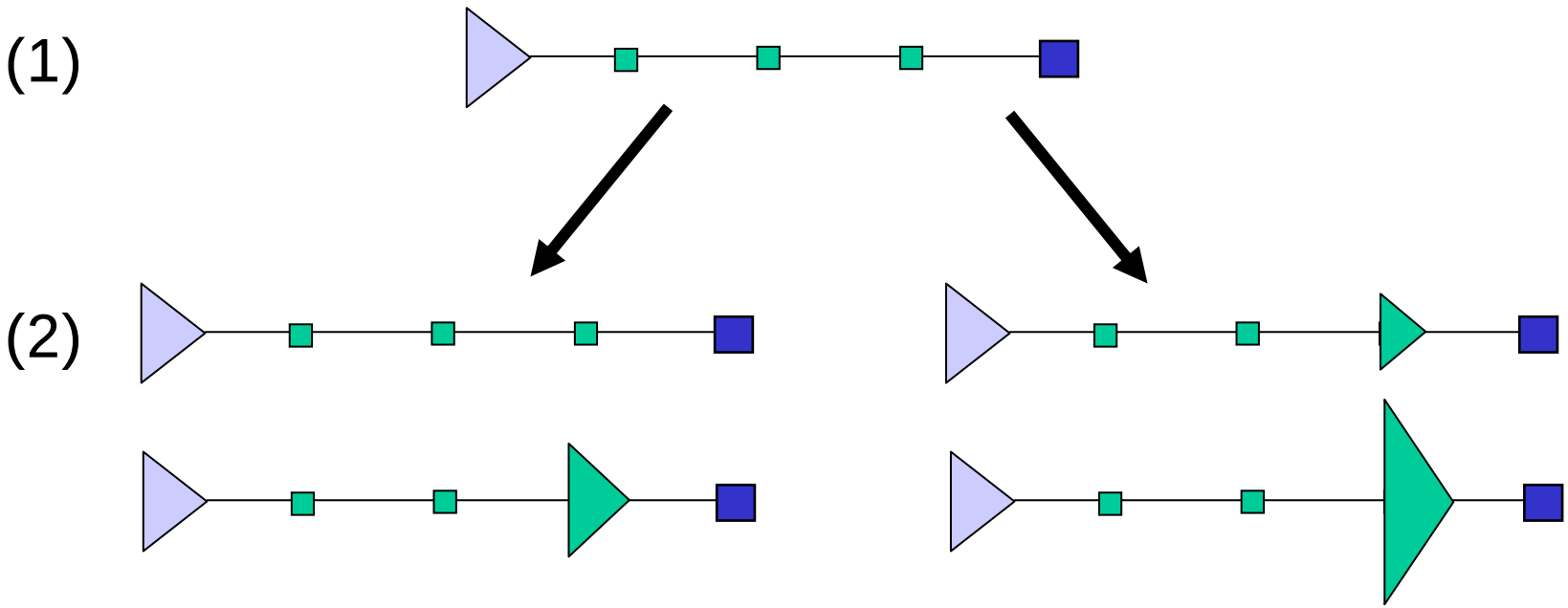
Van Ginneken recap

- Generate candidates from sinks to source
- Quadratic runtime
  - Adding a buffer adds only one new candidate
  - Merging branches additive, not multiplicative
- Optimal for Elmore delay model

Extensions

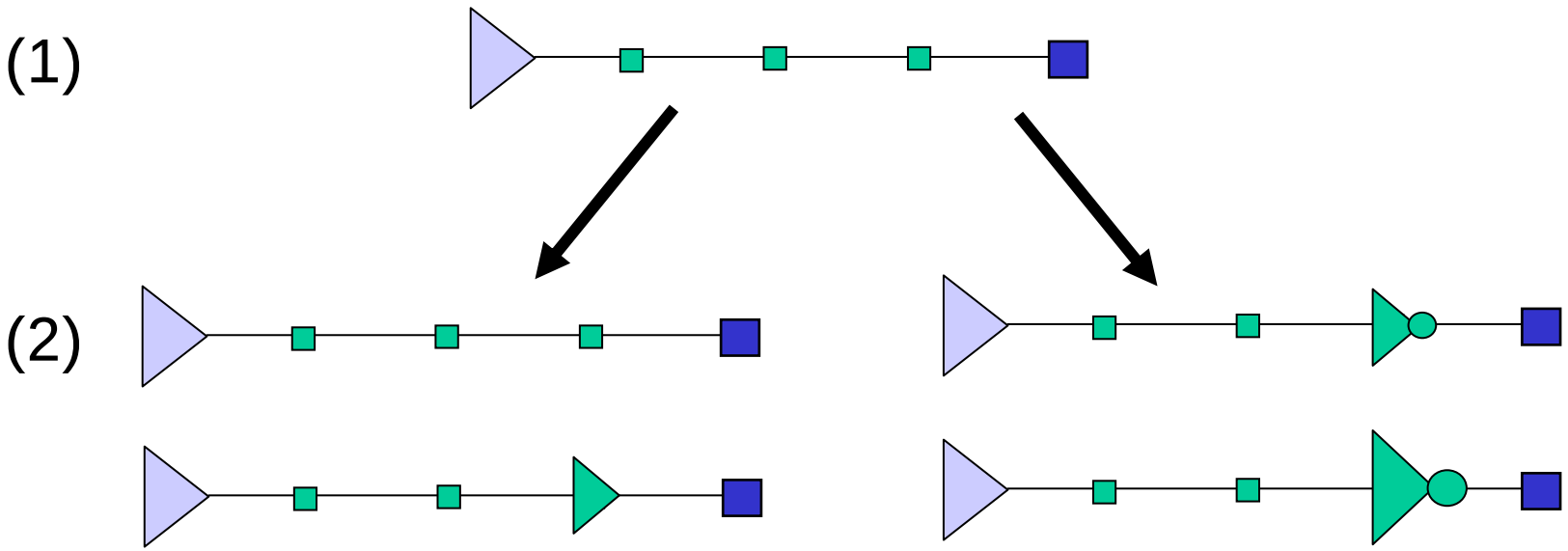
- Multiple buffer types
- Inverters
- Polarity constraints
- Controlling buffer resources
- Capacitance constraints
- Blockage recognition
- Wire sizing

Multiple buffer types



Time complexity increases from $O(n^2)$ to $O(n^2B^2)$ where B is the number of different buffer types

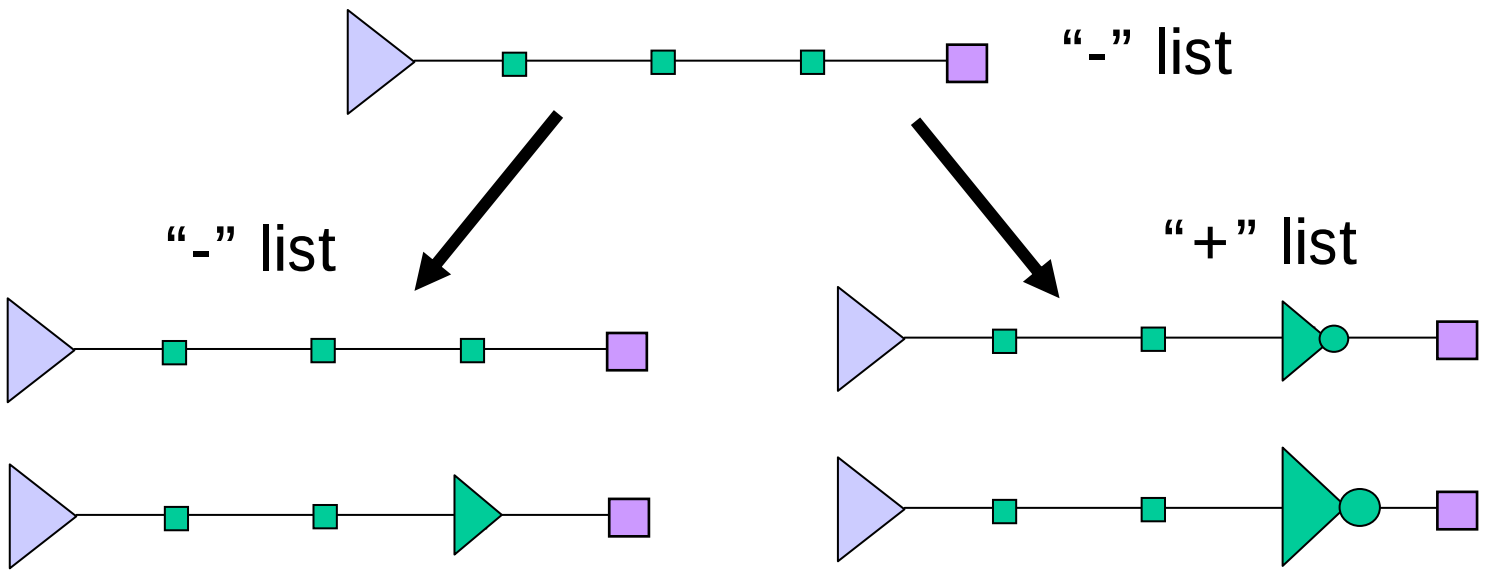
Inverters



- Maintain a “+” and a “-” list of candidates
- Only merge branches with same polarity
- Throw out negative candidates at source

Polarity constraints

- **Some sinks are positive, some negative**
- **Put negative sinks into “-” list**



Controlling buffering resources

Before, maintain list of capacitance slack pairs

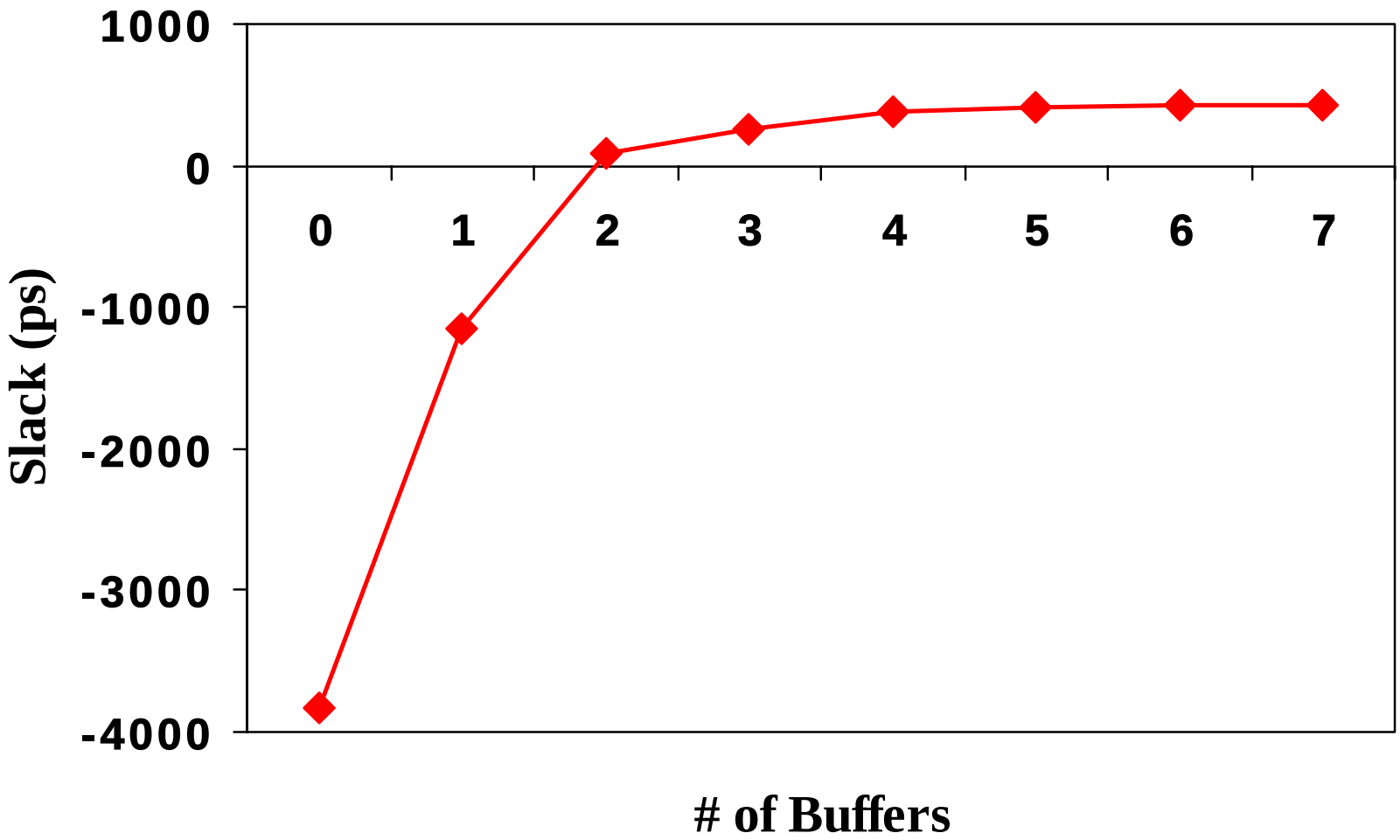
$(C_1, q_1), (C_2, q_2), (C_3, q_3)\ (C_4, q_4), (C_5, q_5)$
$(C_6, q_6), (C_7, q_7), (C_8, q_8)\ (C_9, q_9)$

Now, store an array of lists, indexed by # of buffers

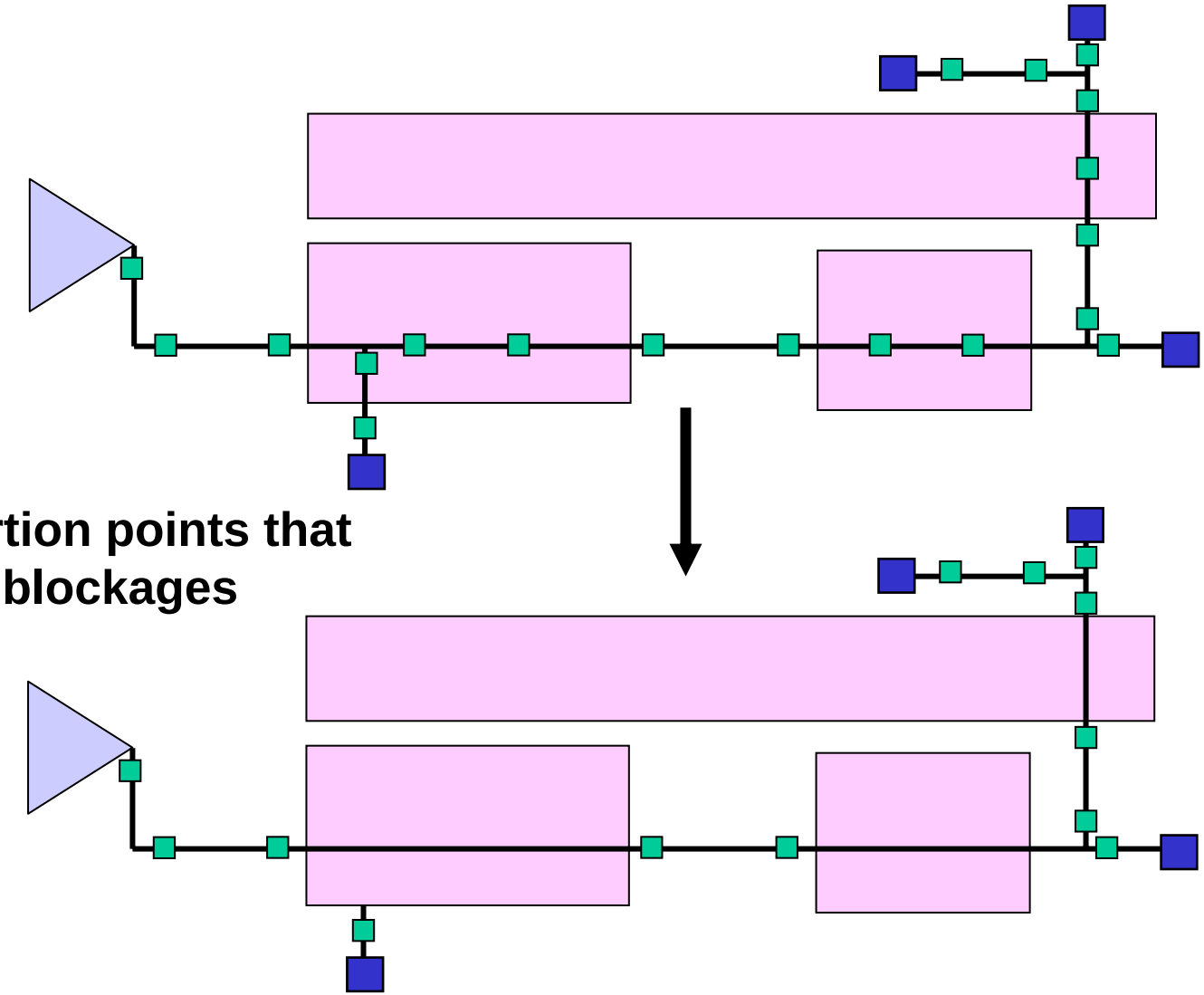
$3 \longrightarrow (C_1, q_1, 3), (C_2, q_2, 3), (C_3, q_3, 3)$
$2 \longrightarrow (C_4, q_4, 2), (C_5, q_5, 2)$
$1 \longrightarrow (C_6, q_6, 1), (C_7, q_7, 1), (C_8, q_8, 1)$
$0 \longrightarrow (C_9, q_9, 0)$

Prune candidates with inferior cap, slack, and #buffers

Buffering resource trade-off



Blockage recognition



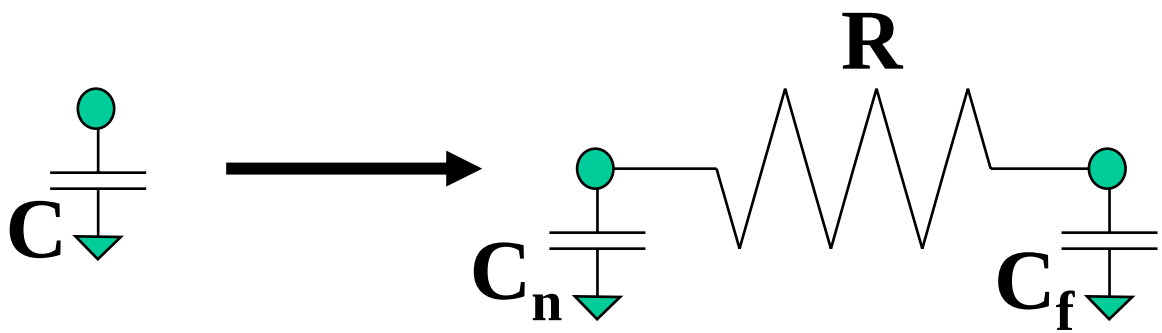
Delete insertion points that run over blockages

Other extensions

- Modeling effective capacitance
- Higher-order interconnect delay
- Slew constraints
- Noise constraints

π-models

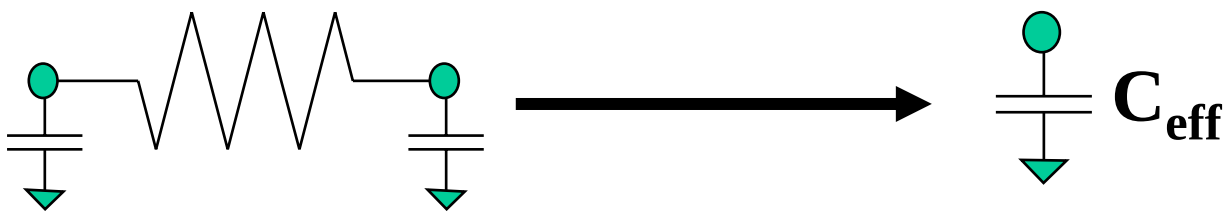
- **Van Ginneken candidate:** (Cap, slack)



- Replace Cap with π-model (C_n, R, C_f)
- Total capacitance preserved: $C_n + C_f = C$
- R represents degree of resistive shielding

Computing gate delay

- When inserting buffer, compute effective capacitance from π-model



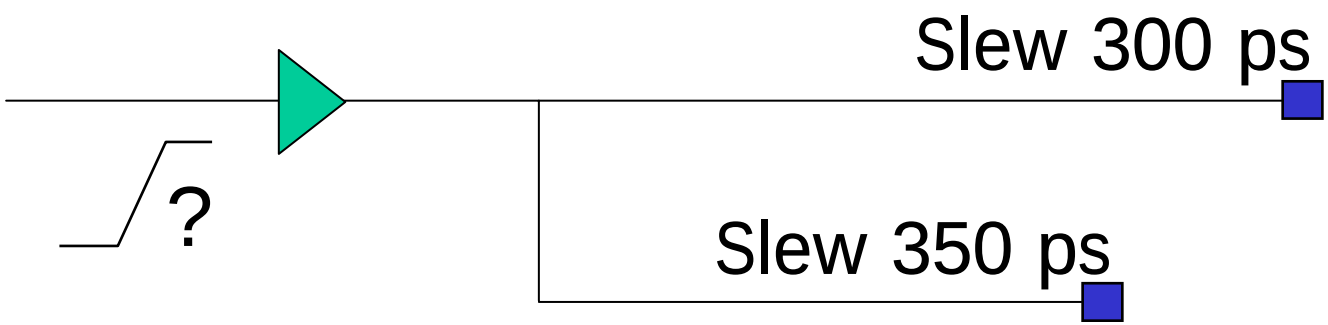
- Use effective instead of lumped capacitance in gate delay equation
- Optimality no longer guaranteed

Higher-order interconnect delay

- Moment matching with first 3 moments
- Previously: candidate (π-model, slack)
- Now: candidate (π-model, m_1, m_2, m_3)
- Given moments, compute slack on the fly
- Bottom-up, efficient moment computation
- Problem: guess slew rate

Slew constraints

- When inserting buffer, compute slews to gates driven by buffer
- If slew exceeds target, prune candidate
- Difficulty: unknown gate input slew



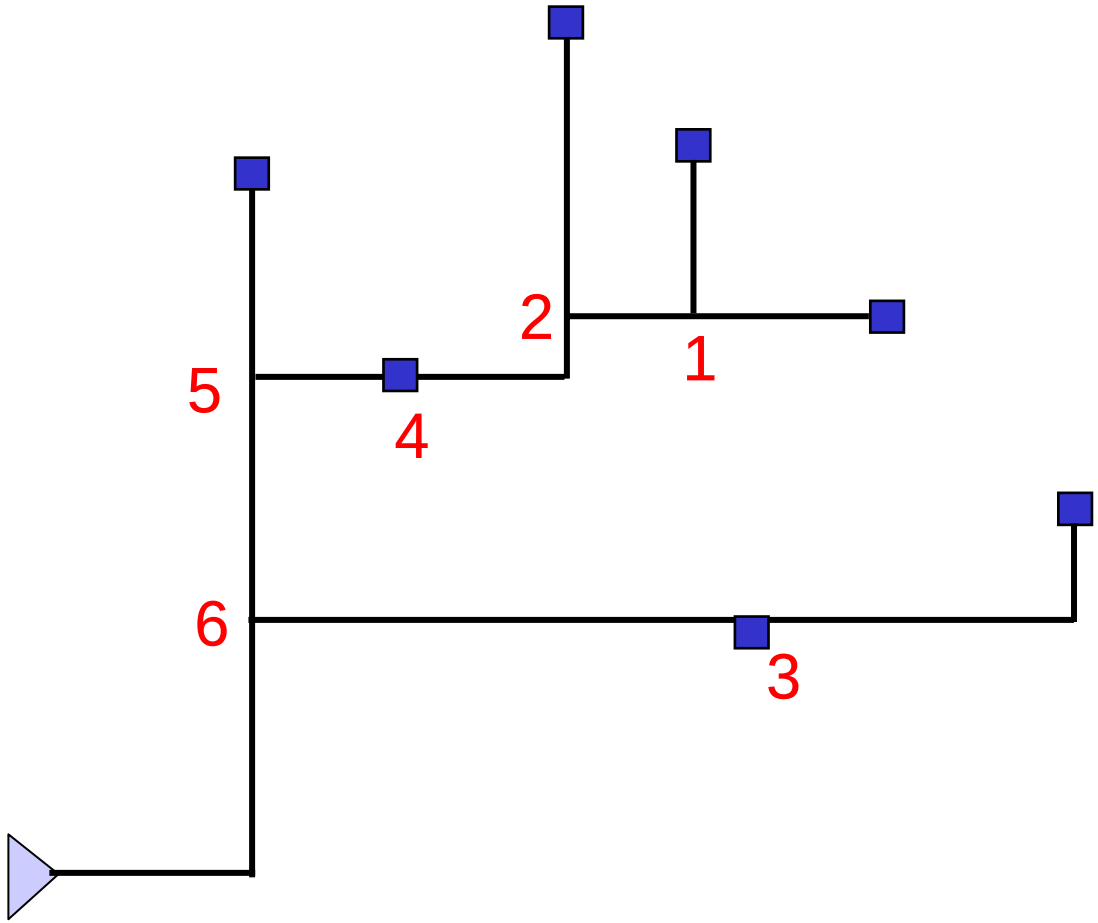
Timing-driven Steiner approaches

- BRBC
- Prim-Dijkstra
- P-Tree
- A-Tree (RSA)
- SERT
- MVERT

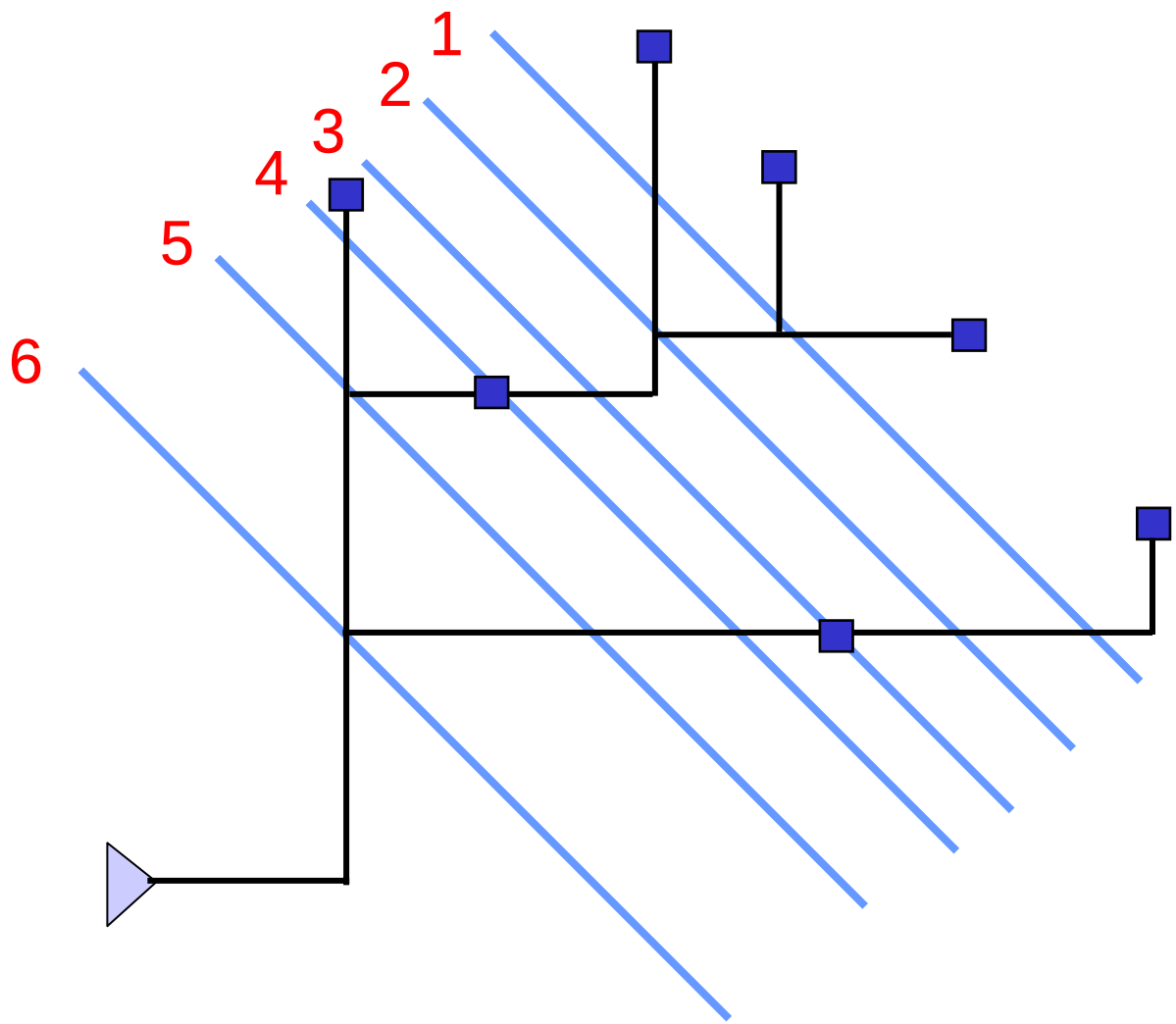
Rectilinear Steiner arborescence

- Assume all sinks in first quadrant
- Iteratively
  - Find sink pair p and q maximimizing $\min(x_p, x_q) + \min(y_p, y_q)$
  - Remove p and q from consideration
  - Replace with $r = \min(x_p, x_q), \min(y_p, y_q)$
  - Connect p and q to r

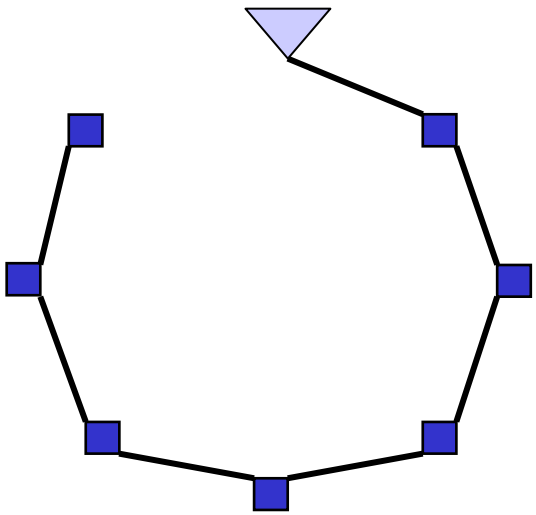
RSA example



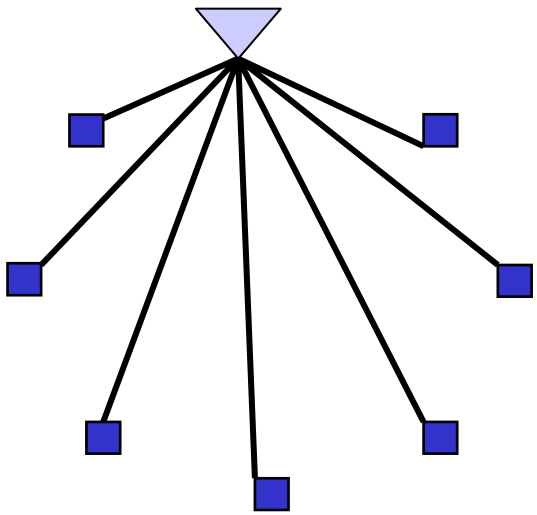
RSA diagonal line sweep



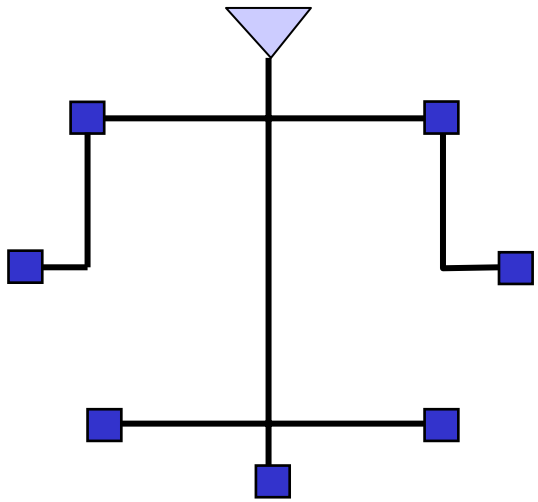
Prim-Dijkstra algorithm



Prim's
MST



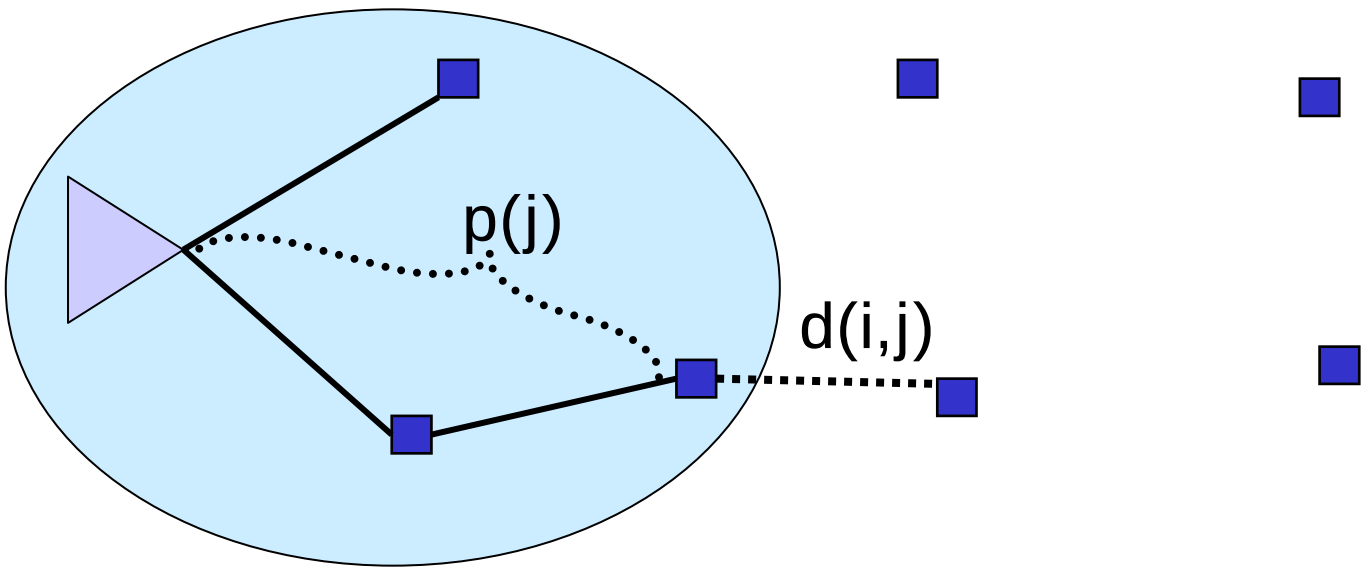
Dijkstra's
SPT



Trade-off

Prim's and Dijkstra's algorithms

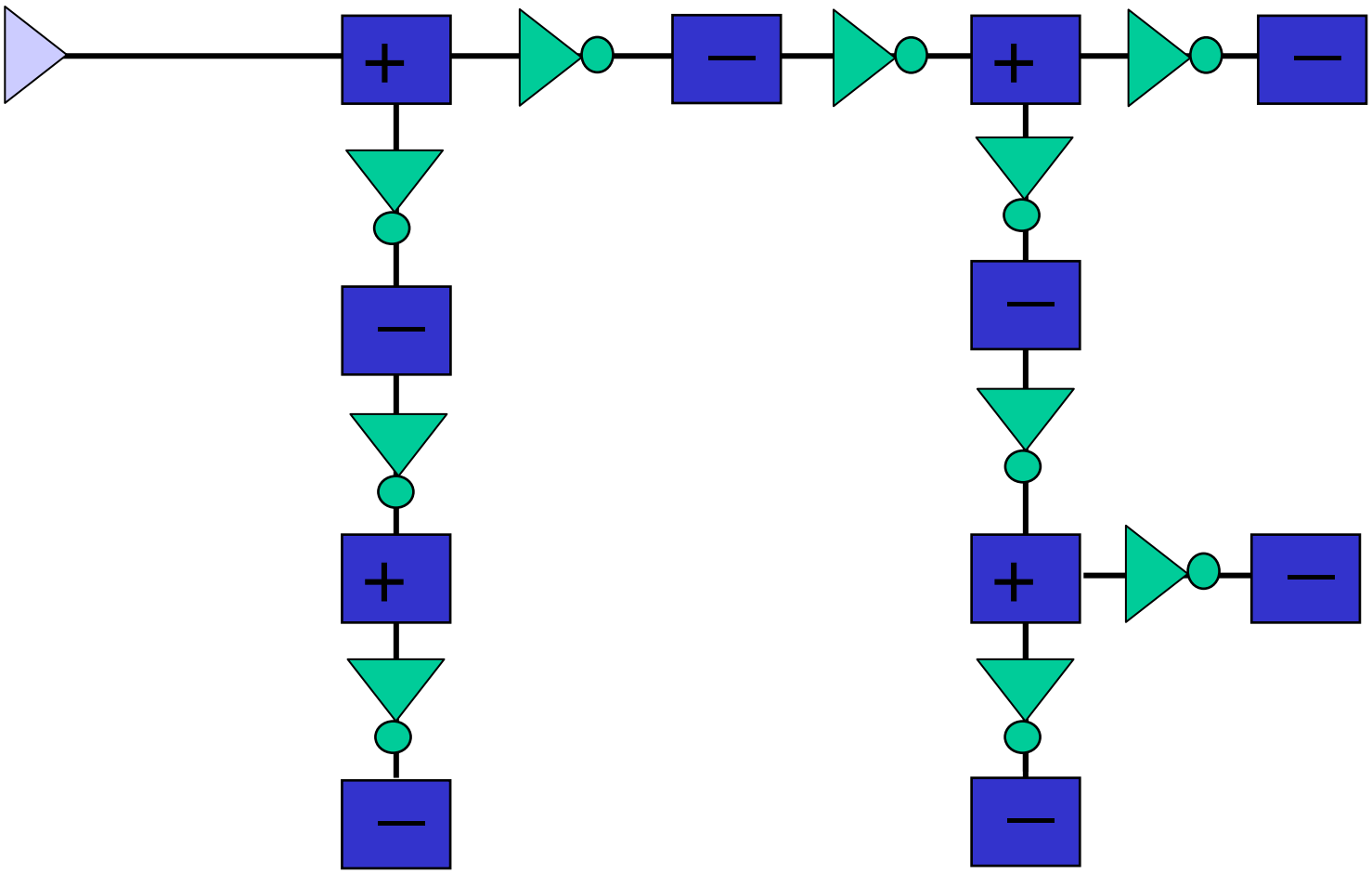
- d(i,j): length of the edge (i, j)
- p(j): length of the path from source to j
- Prim: d(i,j) Dijkstra: d(i,j) + p(j)



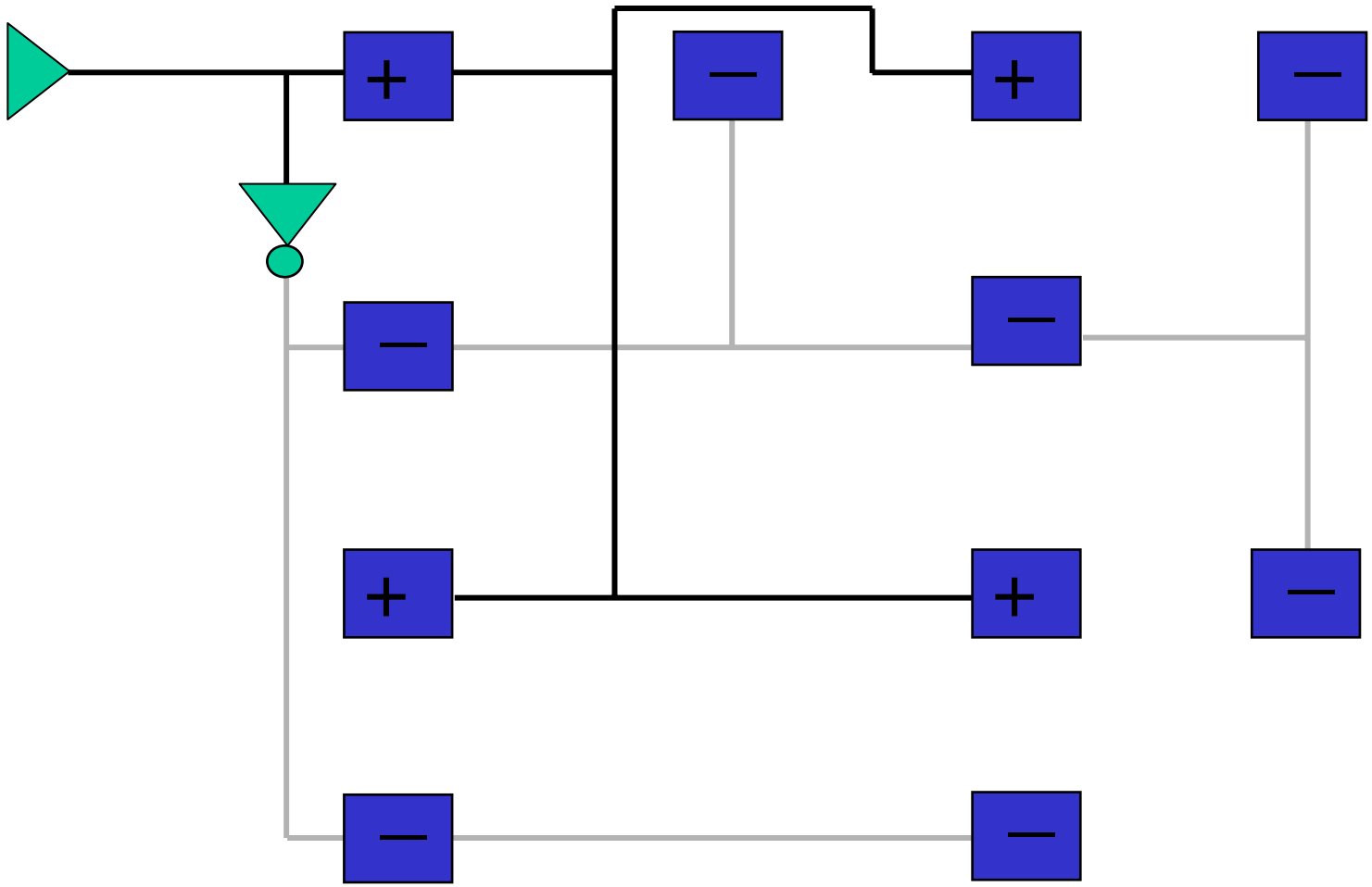
The Prim-Dijkstra trade-off

- Prim: add edge minimizing d(i,j)
- Dijkstra: add edge minimizing p(i) + d(i,j)
- Trade-off: c(p(i)) + d(i,j) for 0 <= c <= 1
- When c=0, trade-off = Prim
- When c=1, trade-off = Dijkstra

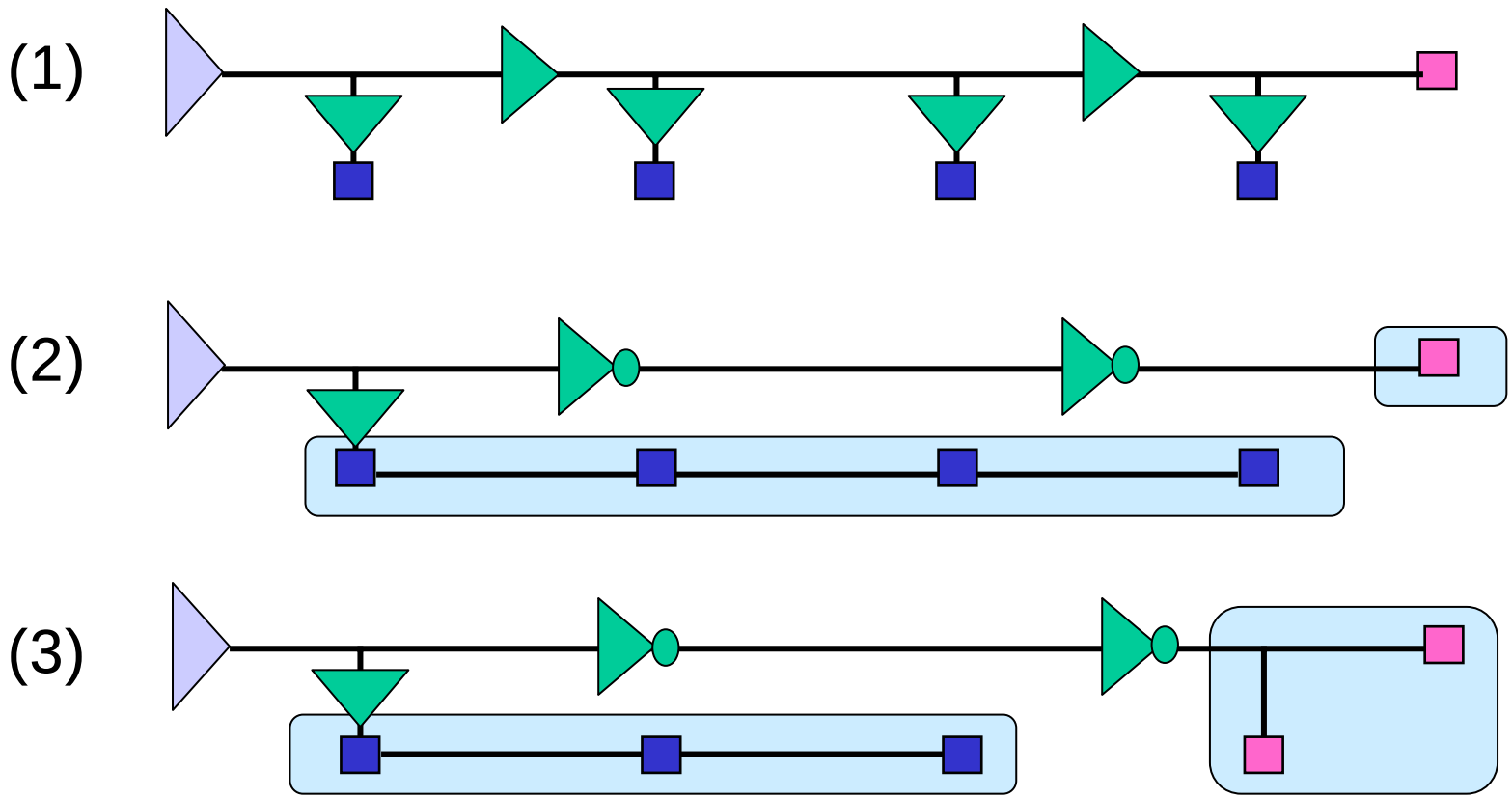
Polarity problem



A better solution?



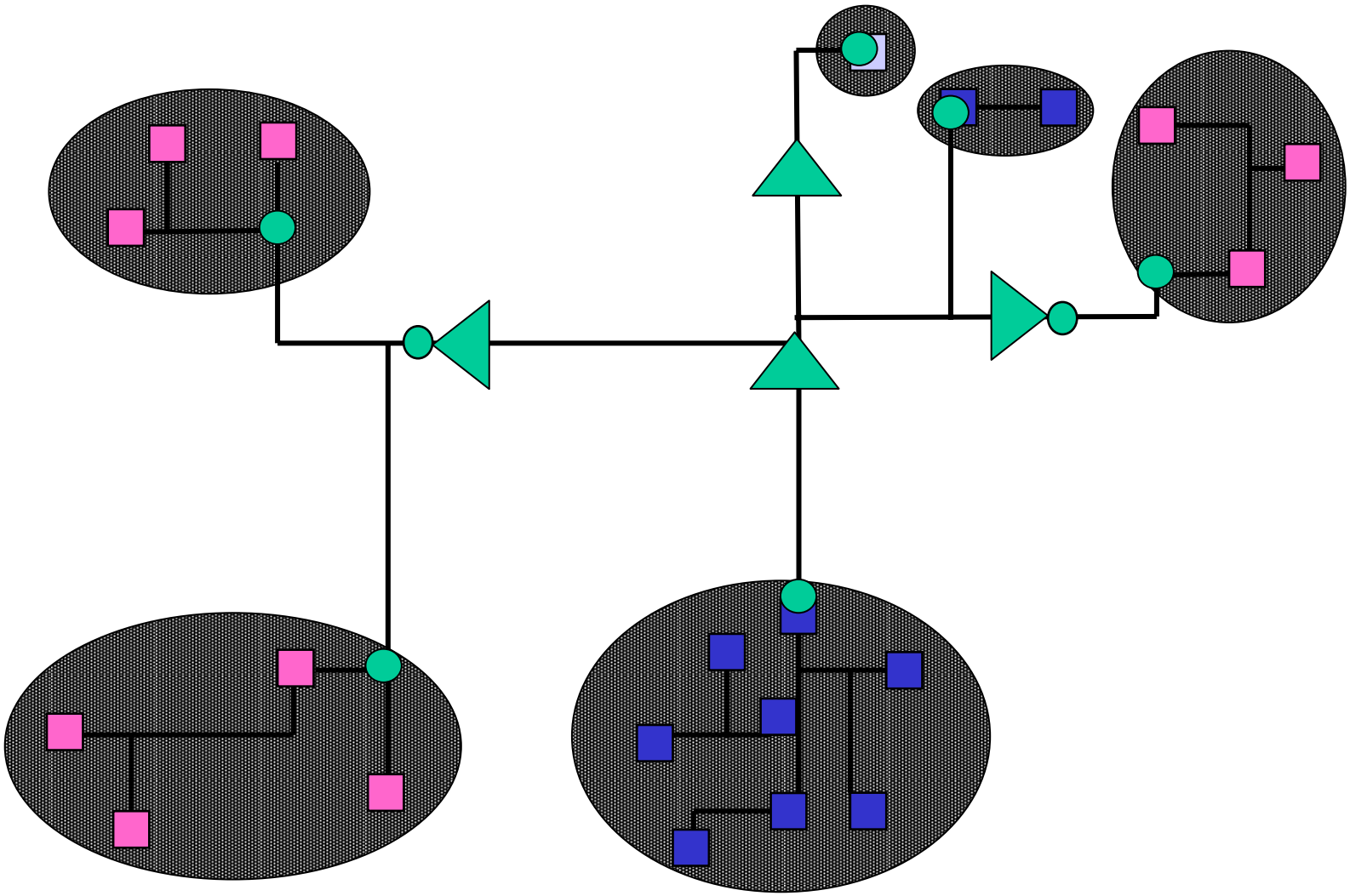
Buffer aware trees



C-Tree algorithm

- Cluster sinks by
  - Polarity
  - Manhattan distance
  - Criticality
- Two-level tree
  - Form tree for each cluster
  - Form top-level tree

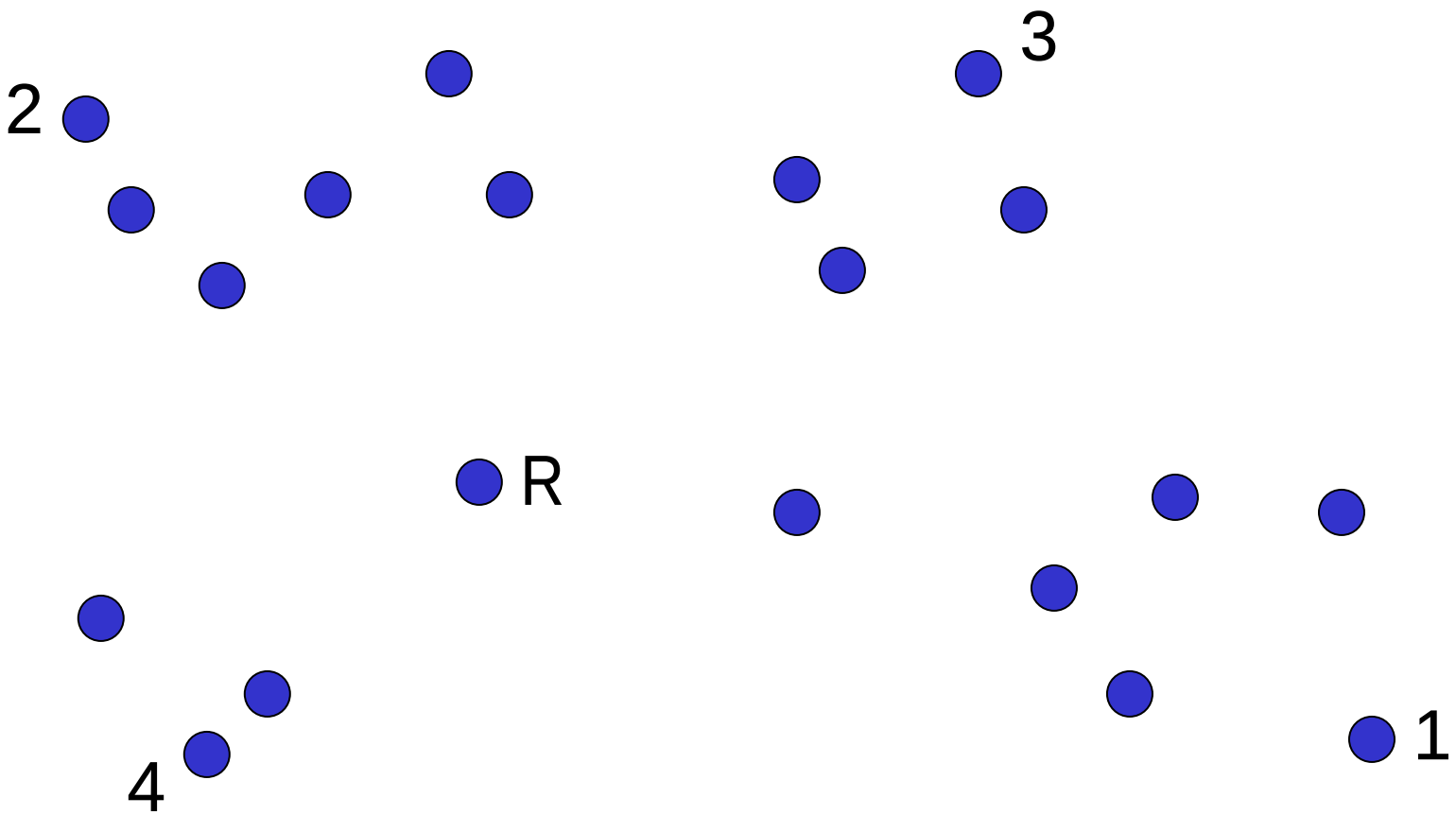
C-Tree example



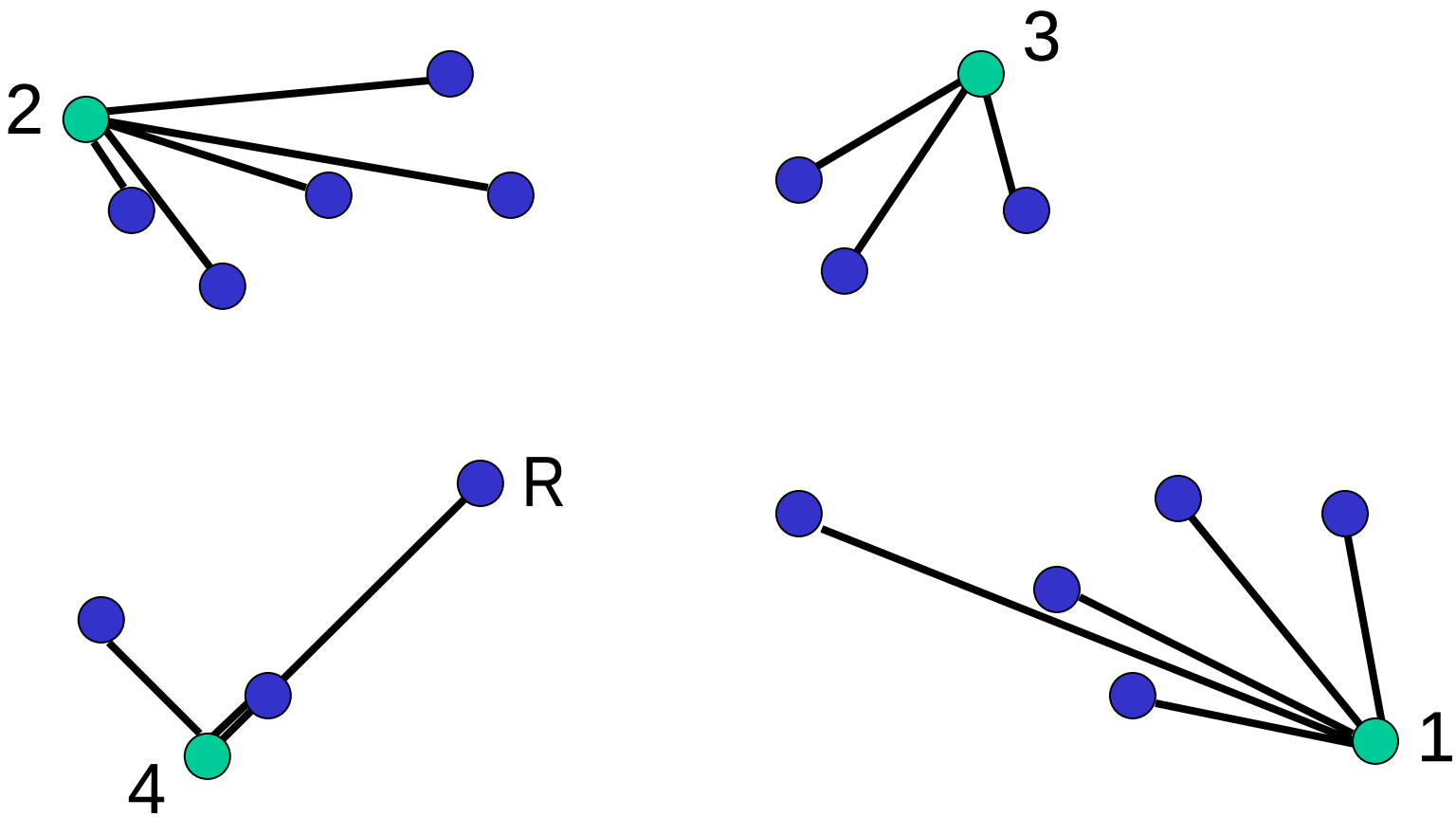
Clustering distance metric

- pDist(i,j) = | polarity(i) – polarity(j)|
- sDist(i,j) = ($|x_i - x_j| + |y_i - y_j|$)/diam
- tDist(i,j) scaled between 0 and 1, 0 for equal criticalities, 1 for opposite criticalities
- Final distance metric d(i,j) = pDist(i,j) + βsDist(i,j) + (1-β)tDist(i,j)

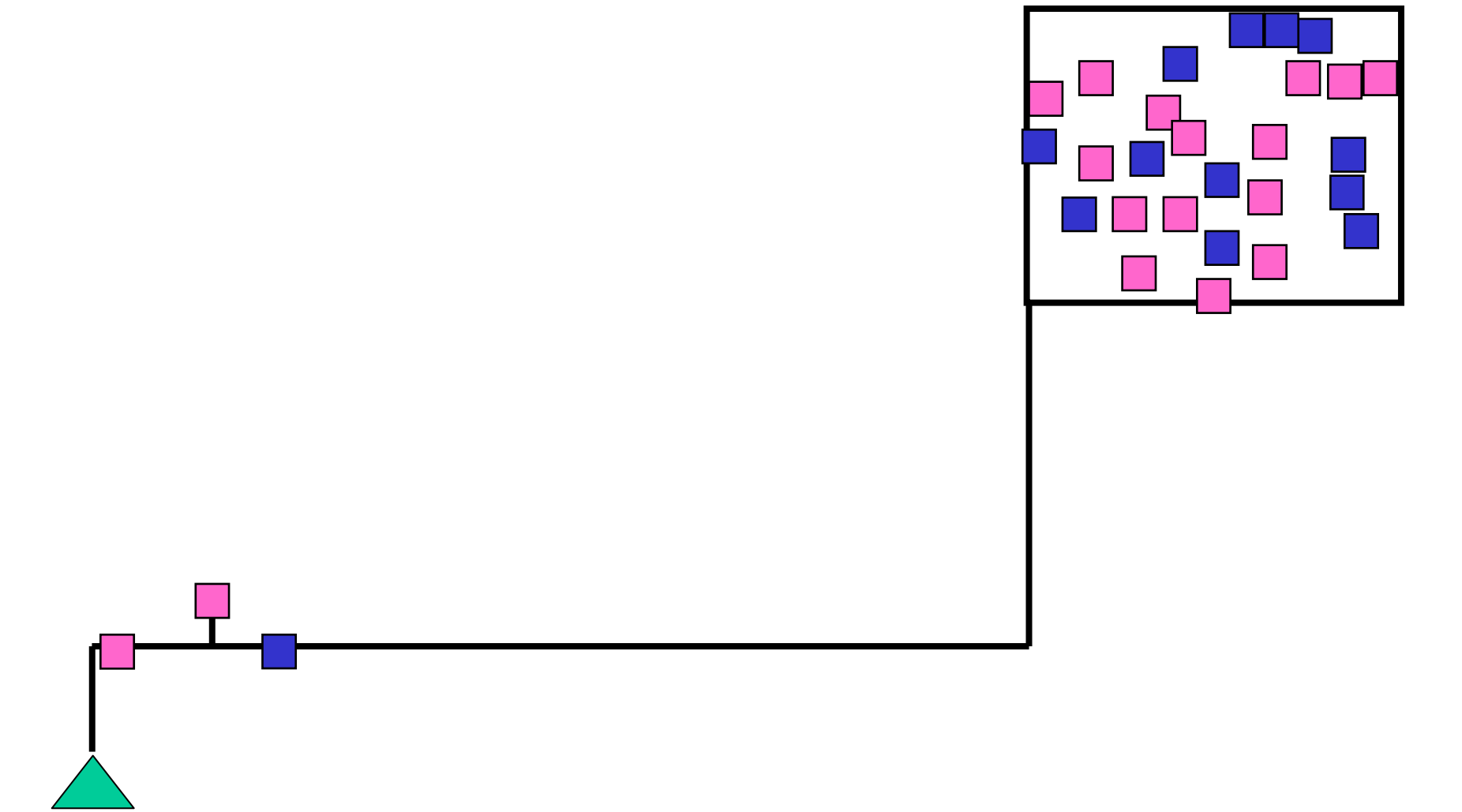
Clustering – Finding centers



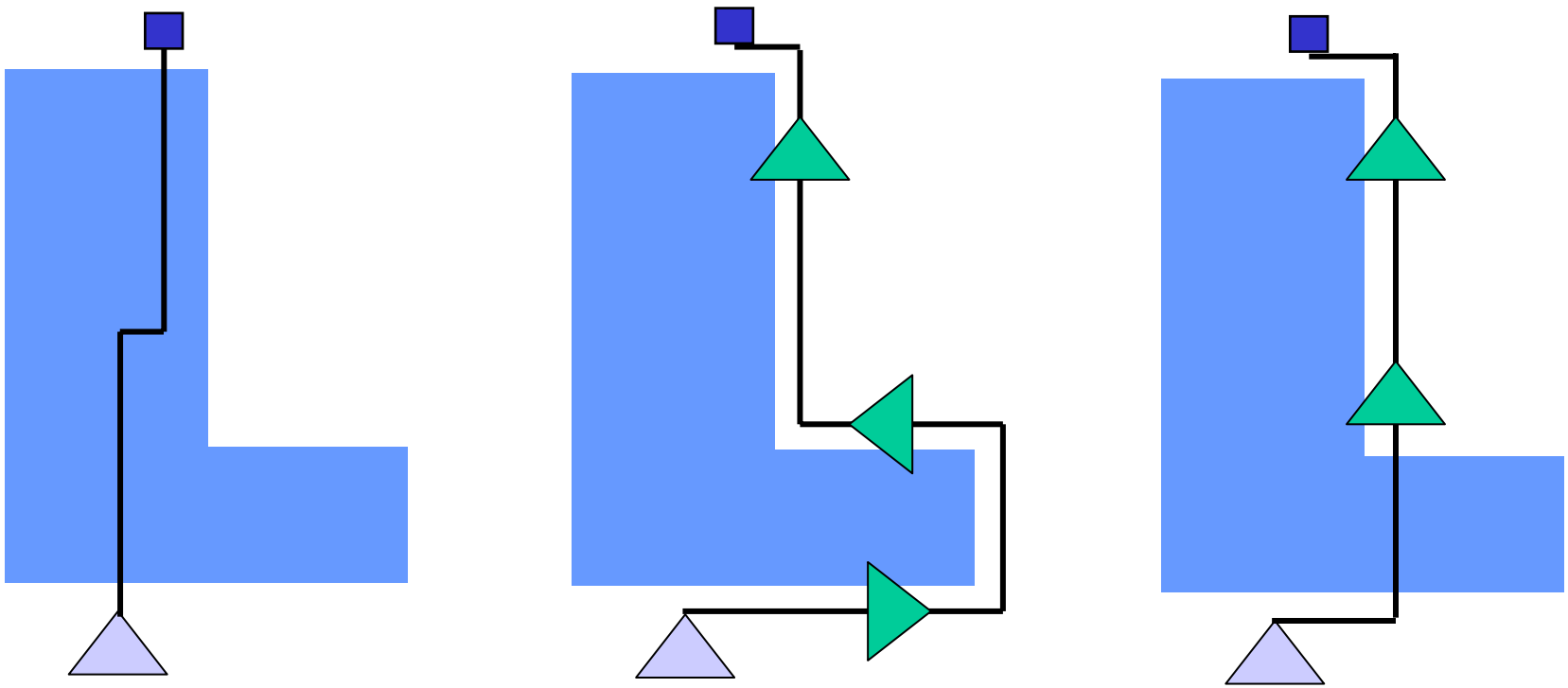
Clustering – Group to centers



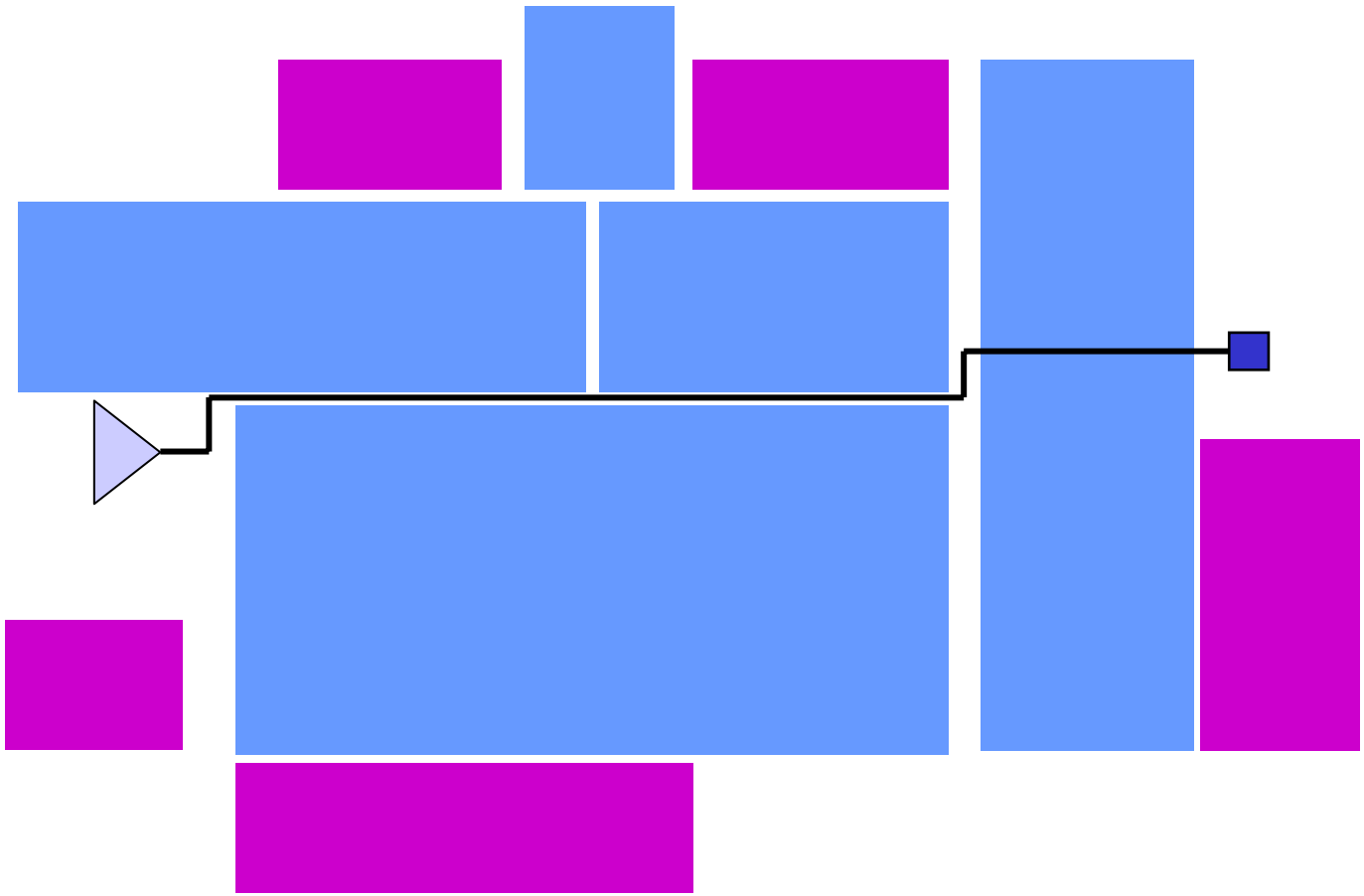
Net n8702



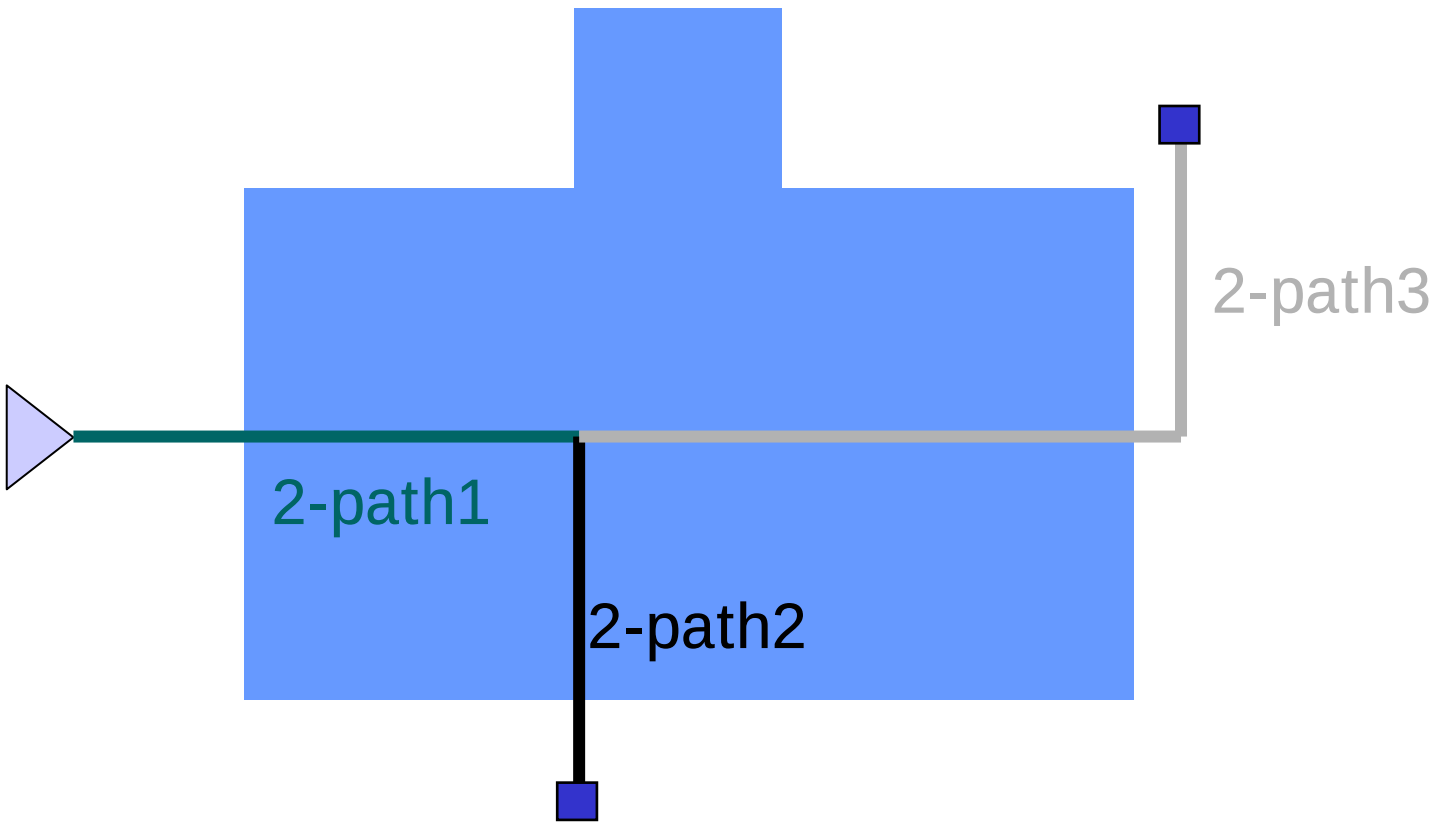
Don’t avoid all blockages!



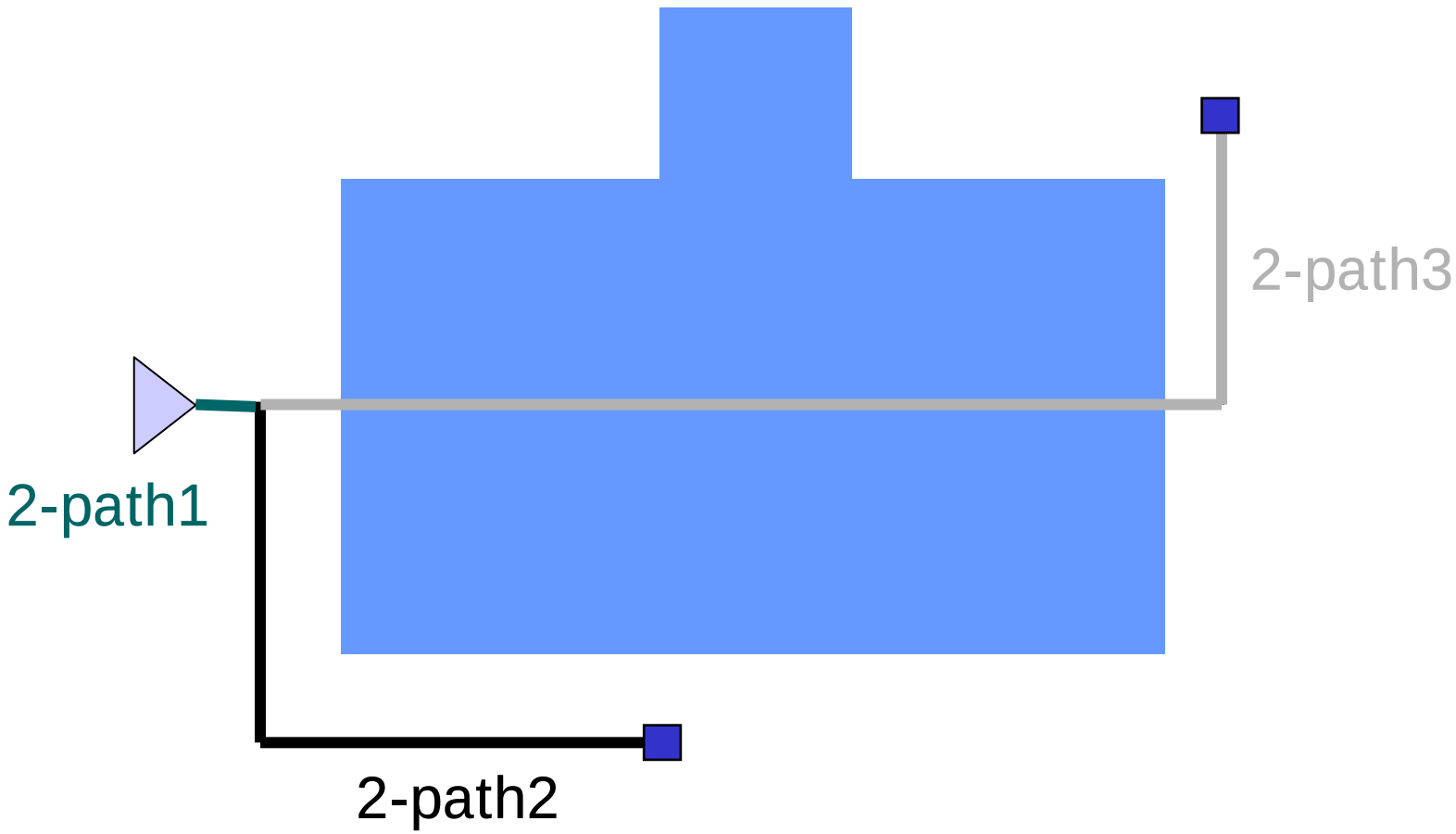
Buffer bays



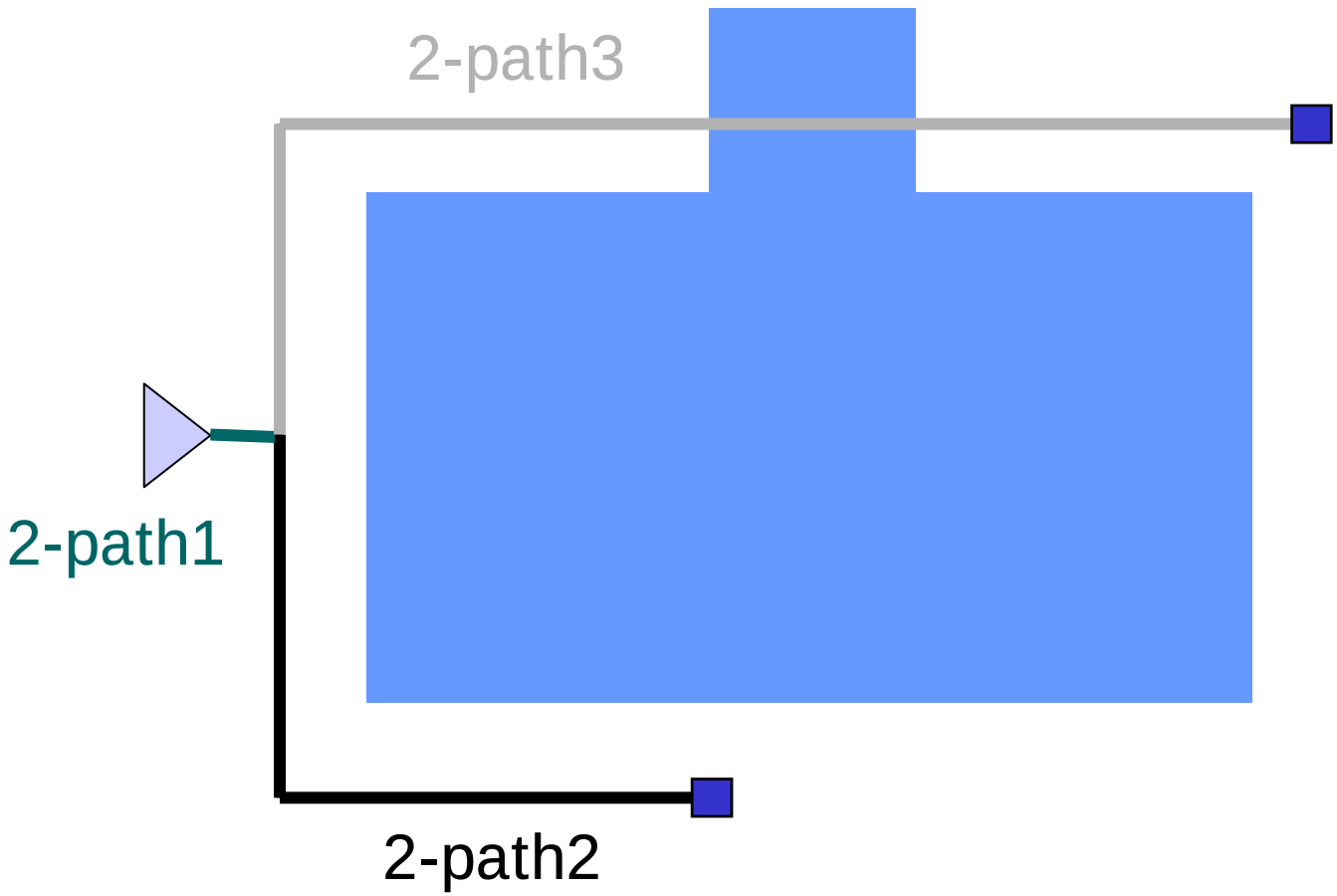
Blockage avoidance example



Blockage avoidance example



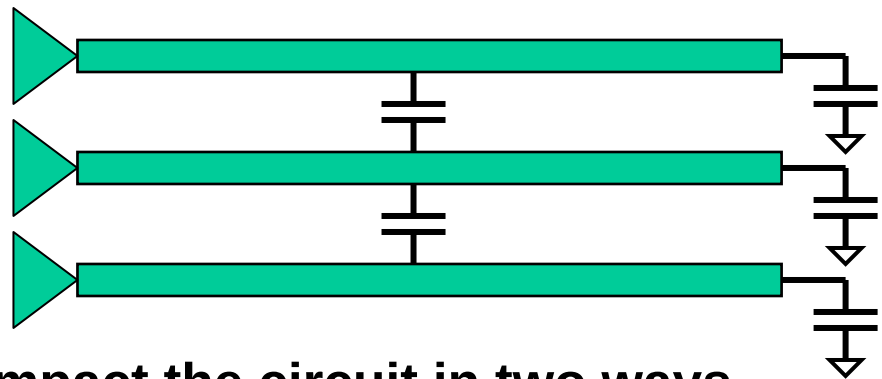
Blockage avoidance example



Noise and Congestion Issues

Crosstalk

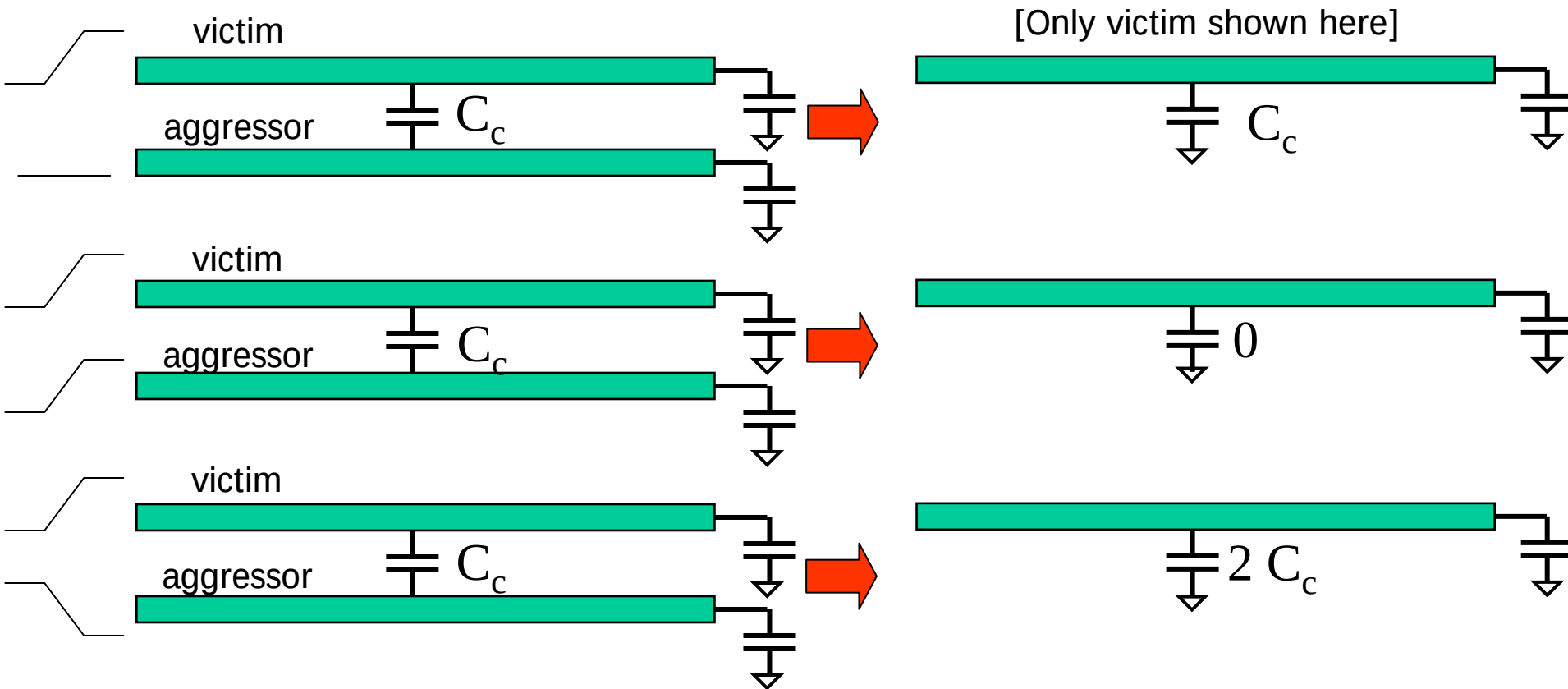
- Crosstalk is caused due to coupling between adjacent wires in a layout
  - Wires have capacitors to GND and between each other
  - $C_{coupling}$ is of the same order of magnitude as $C_{substrate}$



- Coupling can impact the circuit in two ways
  - Increased noise
  - Increased delays
    - "Chicken-and-egg" problem: do not know coupling cap unless delays are known; do not know delays unless coupling cap is known
    - Typically solved by iteration using min-max timing windows

Intuition

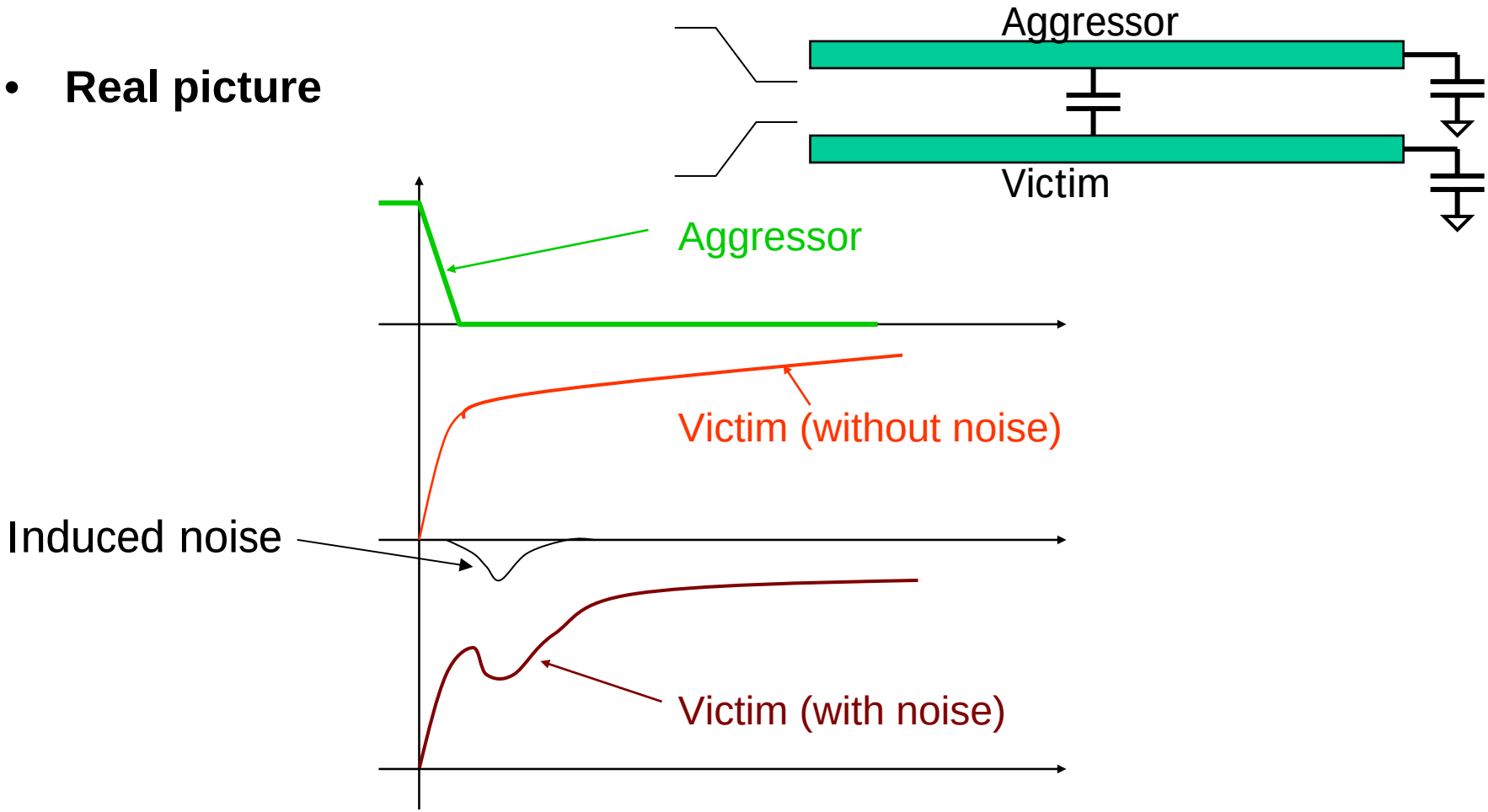
- **Miller capacitance: equivalent capacitor to ground**



  - **In reality, equivalent coupling caps of < 0 and > 2Cc may be seen; use of –C/C/3C has been proposed**

Miller capacitors are an approximation!

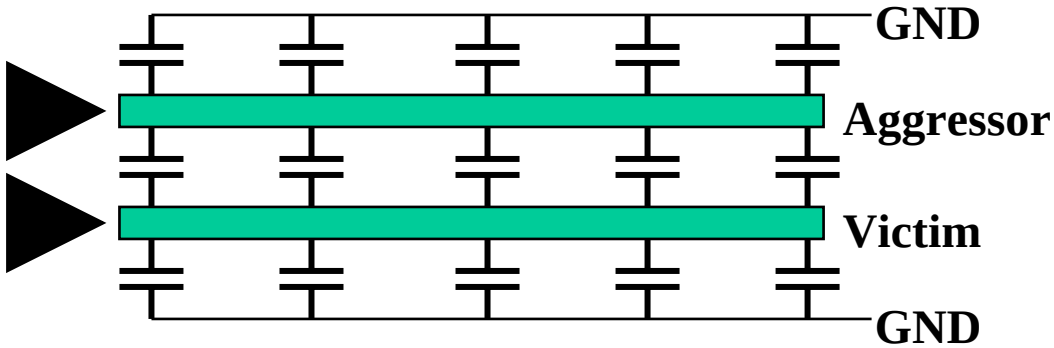
- **Real picture**



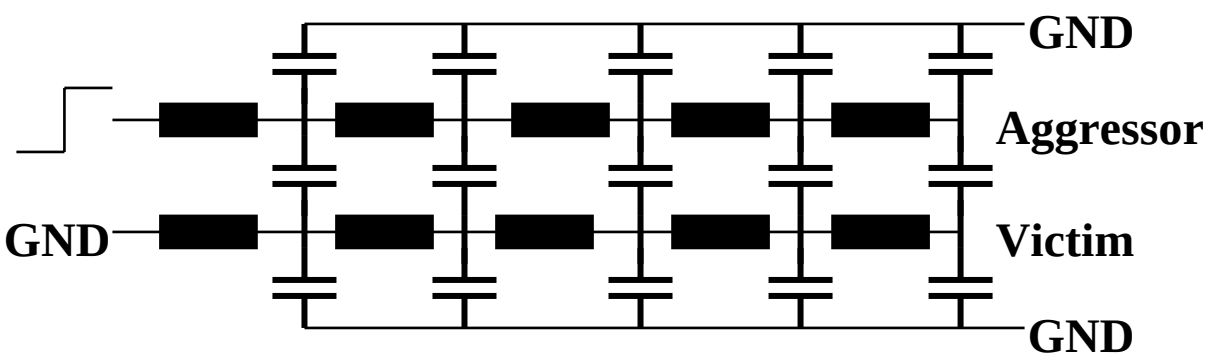
Fanout gate acts as a low-pass filter! If the pulse is very sharp + occurs after the transition, it may be filtered out

Parameters affecting coupling noise

- "Near end" vs. "Far end"

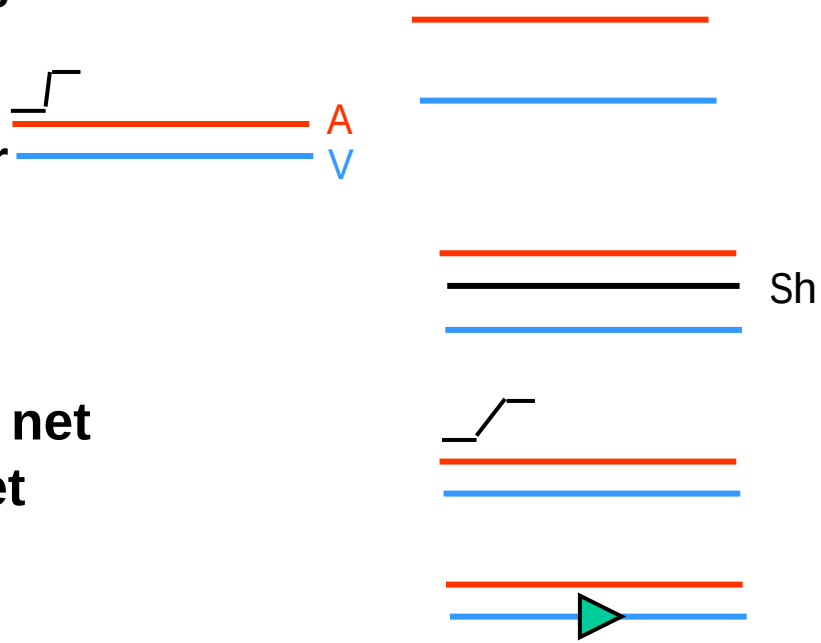


- RC model: $V_{far\ end} > V_{near\ end}$



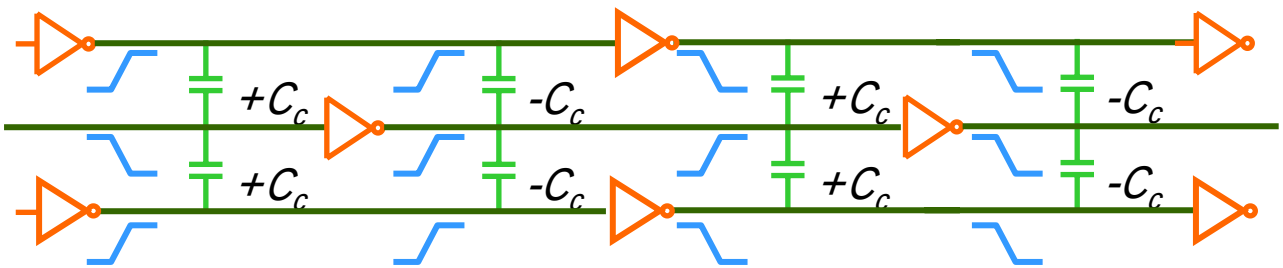
Noise Optimization

- Spacing
- Track permutation
  - Temporally non-adjacent signals made spatially adjacent
- Shielding
- Downsizing aggressor driver
- Upsizing victim driver
- Buffering victim net
- Up-layering victim net
- Changing topology of victim net
- Splitting fanouts of victim net



Bus design

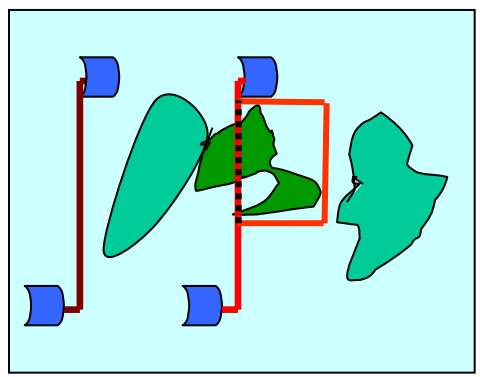
- Bundles of signals treated symmetrically
  - Identical electrical/physical environment for each bit
- Abstraction of communication during early design
  - Often integrated with floorplanning during μarch exploration
- Global busses often pre-designed prior to detailed block implementation (esp. in microprocessors)
- Several speed-up techniques unique to busses
  - Staggered repeaters, swizzling, interleaving of signals traveling in opposite directions
  - Relies on minimizing impact of coupling between adjacent bits



Congestion considerations

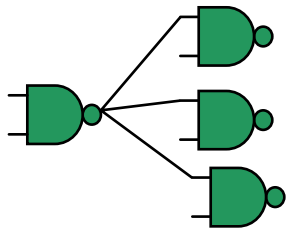
- **Designs increasingly wire-limited**
- **Interconnect optimization: routing resource intensive**
  - **Shielding, spacing, wide-wires, up-layering**
- **Congestion can cause detours (or even unroutable designs)**
- **Detours increase interconnect delay *as well as interconnect delay unpredictability***
  - **Wire delay models during tech-mapping, placement are based on shortest path routing**
  - **Detours increase convergence problems because of poor upstream wire delay modeling**

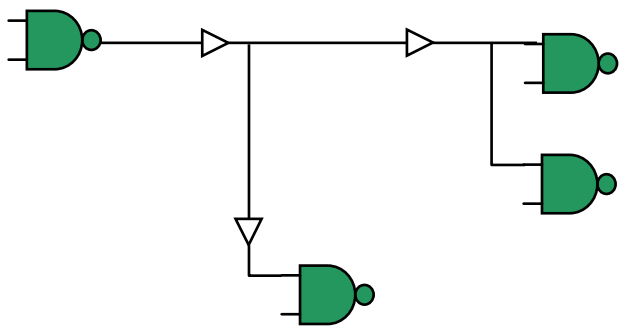
Need to model actual layers, routes for critical nets during placement



Impact on synthesis

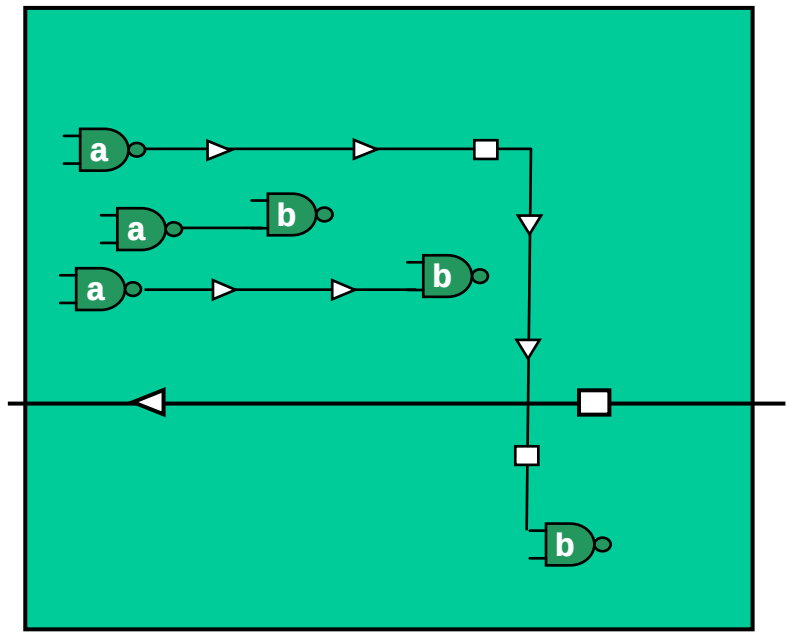
- **Wires cannot be ignored during synthesis**
  - **Fanout based load models obsolete … but wireload models still very inaccurate**
  - **Fanouts often isolated by buffers**
- **Literal/gate count metrics often misleading**
  - **Area is often wire-limited**
  - **Area impact of wire-RC buffers**
- **Pre-layout gate sizing is wasted**
- **Dense encodings (vs. one-hot and other sparse encodings)** ⇩





Buffering and placement

- # buffers needed on a net depends on its routing
- Net routing depends on placement
- Buffer management for intra-block vs global nets
  - Too restrictive to treat global routes/buffers as fixed obstructions



Full-chip assembly issues

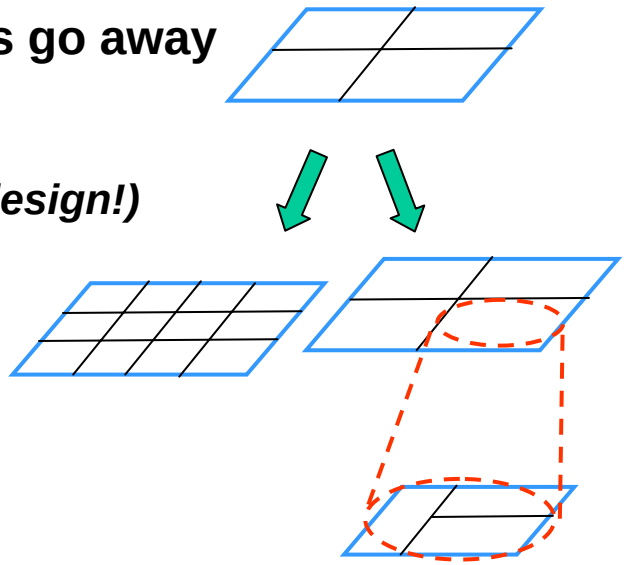
What if we reduce block area to avoid wire effects?

Many of the new physical synthesis problems go away

BUT

 # blocks increases!

(and block assembly is the hardest part of chip design!)



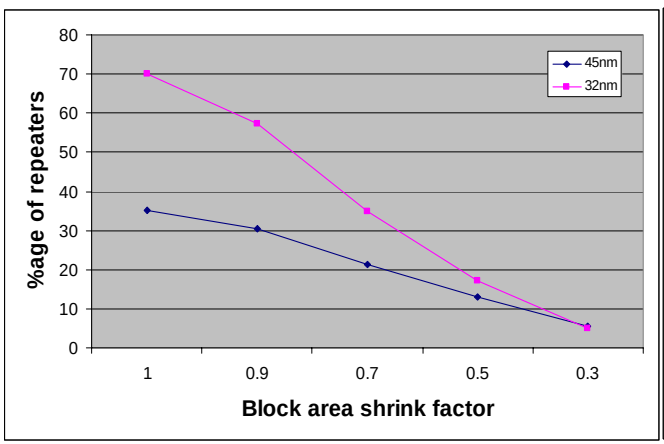
- **Flat assembly**

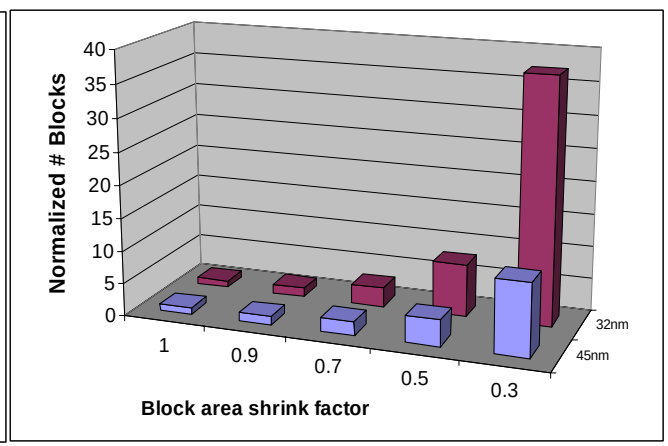
  (Fragmentation of paths across blocks)

  OR

- **Increased hierarchy**

  (Lack of visibility across hierarchy levels)



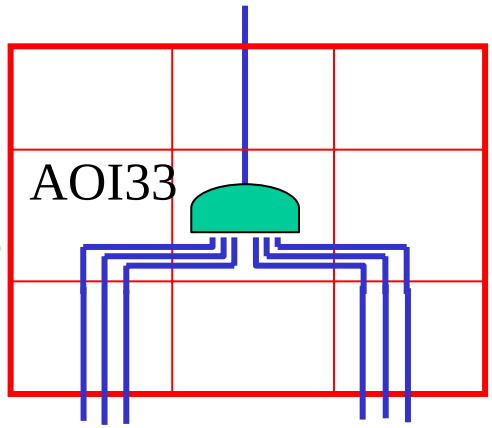


Integrated synthesis and placement

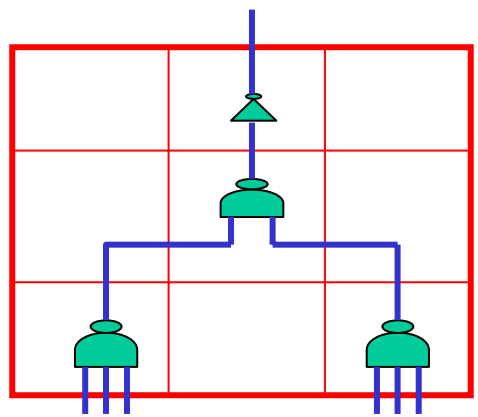
- **Since design metrics depend heavily on layout, generate a layout plan as early as possible**
- **Evolve logic and its layout in tandem ("c*ompanion placement* ")**
  - **Integrate logic synthesis / tech mapping with global placement**
  - **Embed nodes spatially through recursive logic partitioning and placement**
  - **Long, critical wires and buffer needs identified early**
  - **Wire loads obtained using embedding of nodes**
  - ***Hard to estimate area or delay of a Boolean node or FSM***
    - **Pin positions can help**
  - **Somewhat easier at tech mapping stage…**
- **Most industrial physical synthesis tools involve some integration between tech mapping and placement**

Congestion optimization

- Congested layouts harder to converge or unroutable
  - More delay from wires
  - Detours make upstream wire delay models more inaccurate
- Cannot model congestion by a single number characterizing entire block
  - Spatial map required
- Congestion can be addressed during placement
  - Congestion cost in objective function
  - Post-placement remedies
- Recent work on congestion relief by modifying netlist structure during tech mapping
  - Congestion map generated bottom-up during covering from partial maps propagated during matching



Track requirement = 20

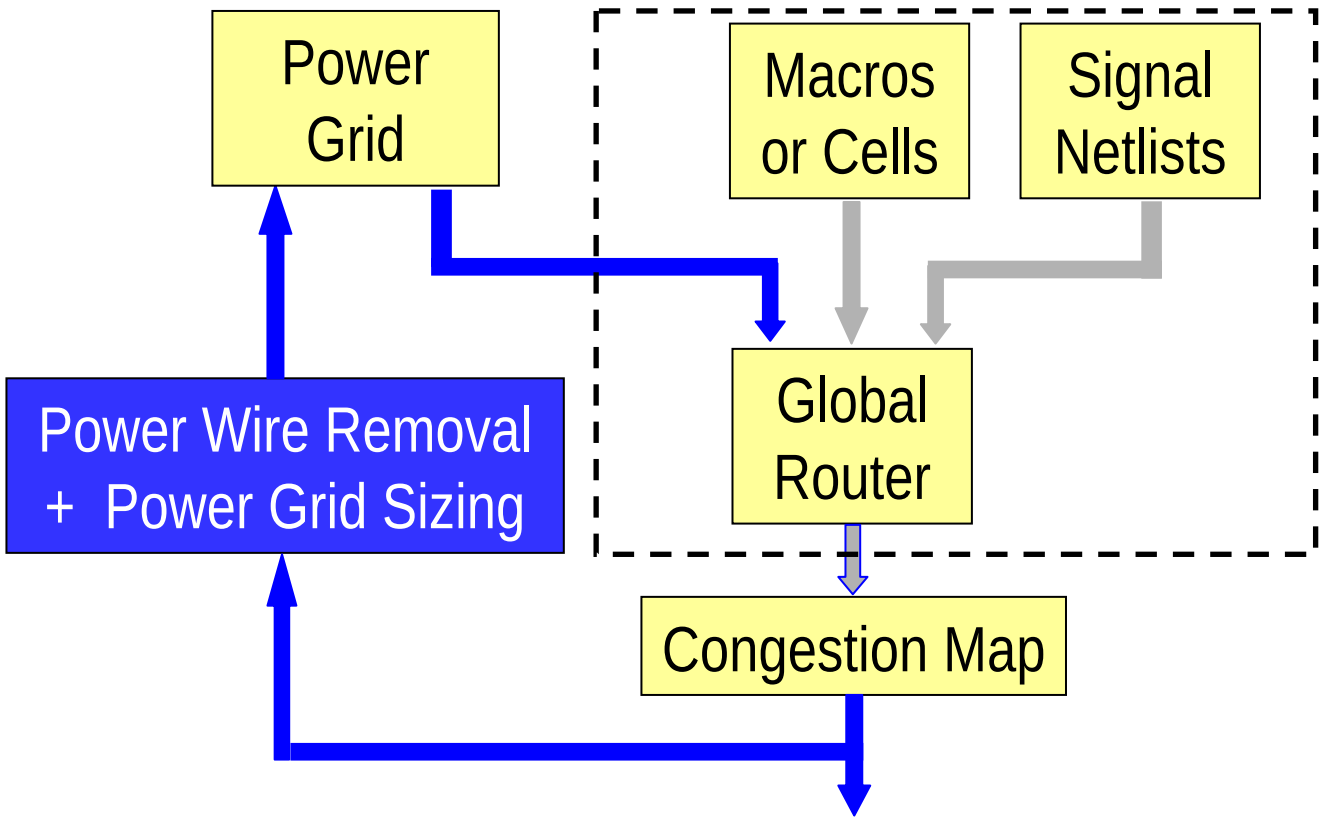


Track requirement = 12

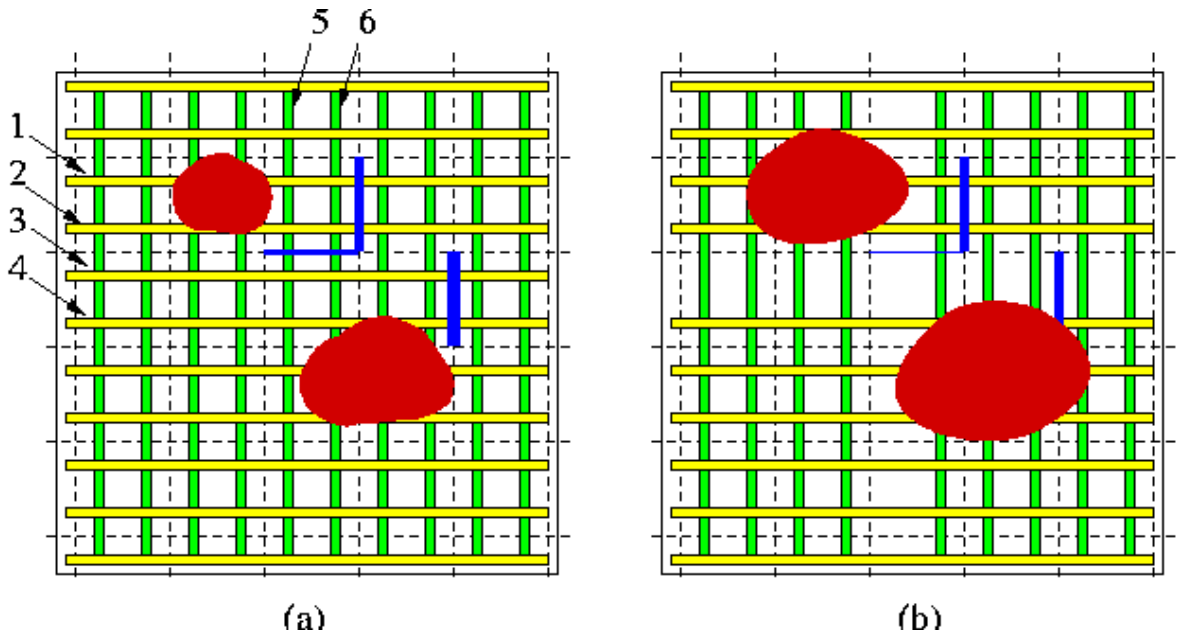
(Shelar, ISPD'05)

Congestion driven supply/signal codesign

- **Interconnect resources increasingly scarce**
  - **Global power and signal wires compete for routing resources**



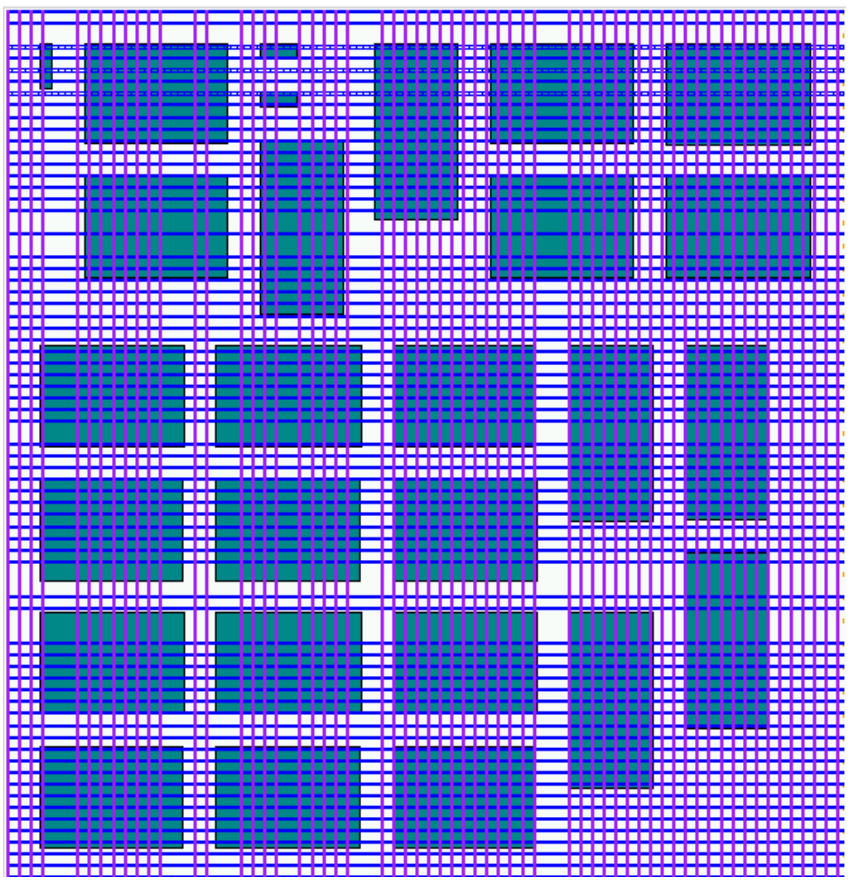
Removal illustration

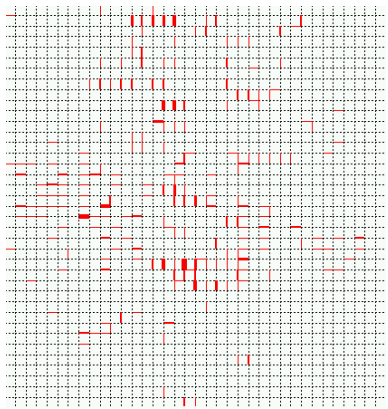


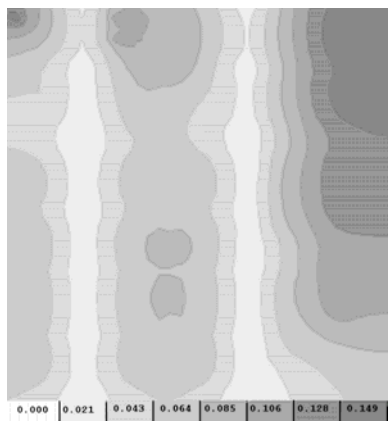
Critical wires: 1, 2, 4 and 6
Non-Critical wires: 3 and 5

Removal order: first 3 then 5

Optimal power grid of “ac3”







Conclusion

- Interconnects are the primary bottleneck in design today
- Many shifts in design methodology can be motivated by interconnect-related problems (including async or NoCs)
- The objective of this tutorial was to
  - explain why interconnects are important
  - overview some fundamental algorithms in interconnect design
  - outline issues that a designer must worry about